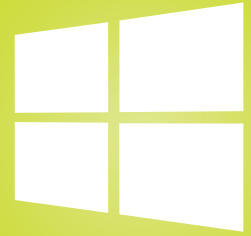


msdn magazine



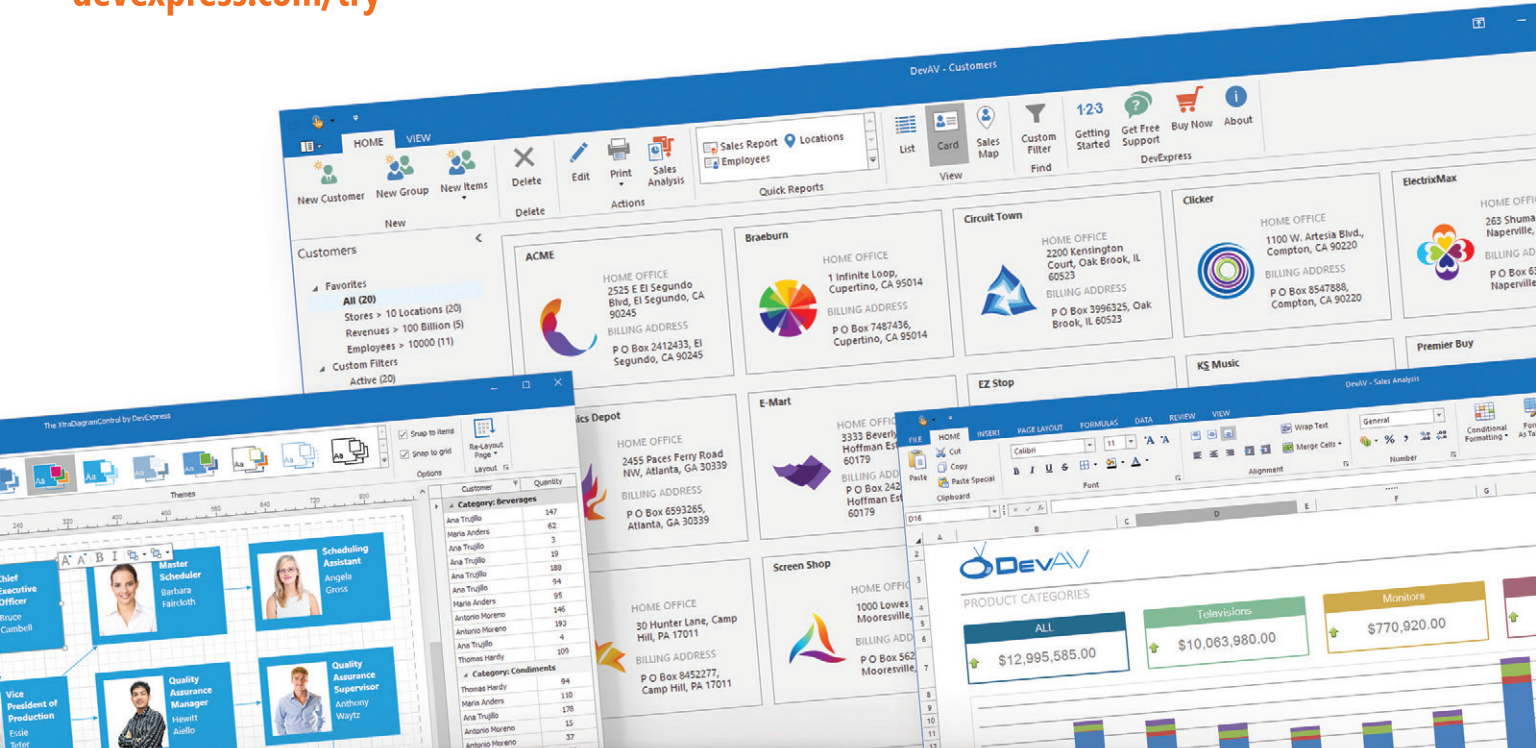
Universal Windows Platform
Development.....10, 20



The King of Desktop Development

Your peers have voted DevExpress Desktop Components best-in-class for 6 straight years. We invite you to see why. Download your free 30-day trial today.

devexpress.com/try





Your Next Great Desktop App Starts Here

From apps that replicate the look and feel of Microsoft Office® to high-powered data mining and decision support systems for your enterprise, DevExpress desktop controls will help you build your best, without limits or compromise.

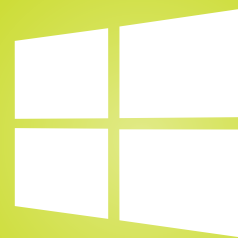


Download your free 30-day trial
and experience the DevExpress difference today.

devexpress.com/try

msdn

magazine



Universal Windows Platform
Development.....10, 20

Closing UWP-Win32 Gaps Andrew Whitechapel	10
Building Connected Apps with UWP and Project Rome Tony Champion	20
Detect and Respond to Rooted Android Devices from Xamarin Apps Joe Sewell	34
Any Language, Any Platform with Azure DevOps Projects Willy-Peter Schaub and Alex Mullans	42

COLUMNS

EDITOR'S NOTE

Improving UWP
Michael Desmond, 4

ARTIFICIALLY INTELLIGENT

Text Sentiment Analysis
Frank La Vigne, page 6

CUTTING EDGE

Under the Covers of
ASP.NET Core SignalR
Dino Esposito, page 50

DON'T GET ME STARTED

Overmind
David Platt, page 56

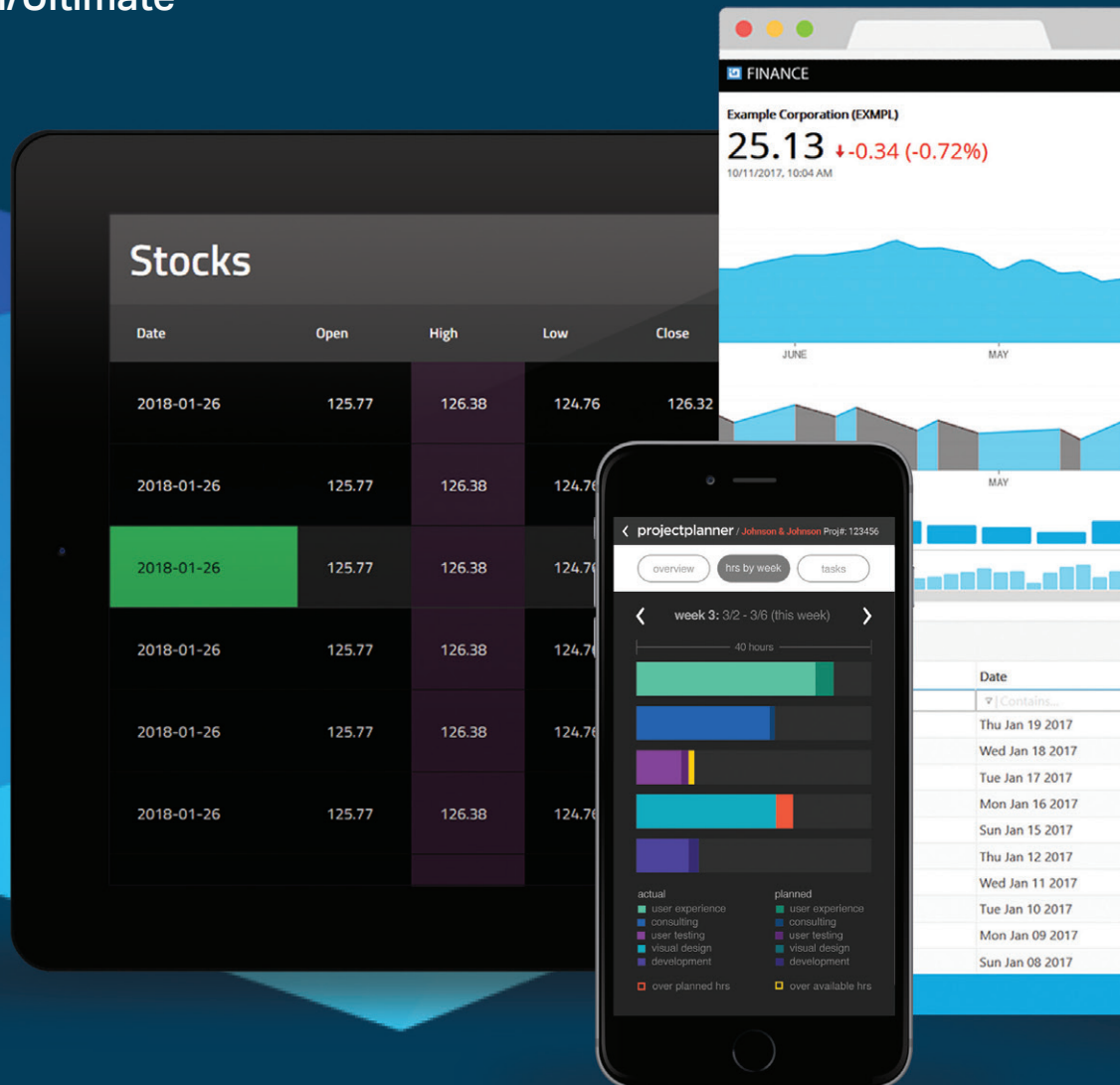


Faster Paths to Amazing Experiences

Infragistics Ultimate includes 100+ beautifully styled, high performance grids, charts & other UI controls, plus productivity tools for building web, desktop and mobile apps.

Angular | JavaScript / HTML5 | ASP.NET | Windows Forms | WPF | Xamarin

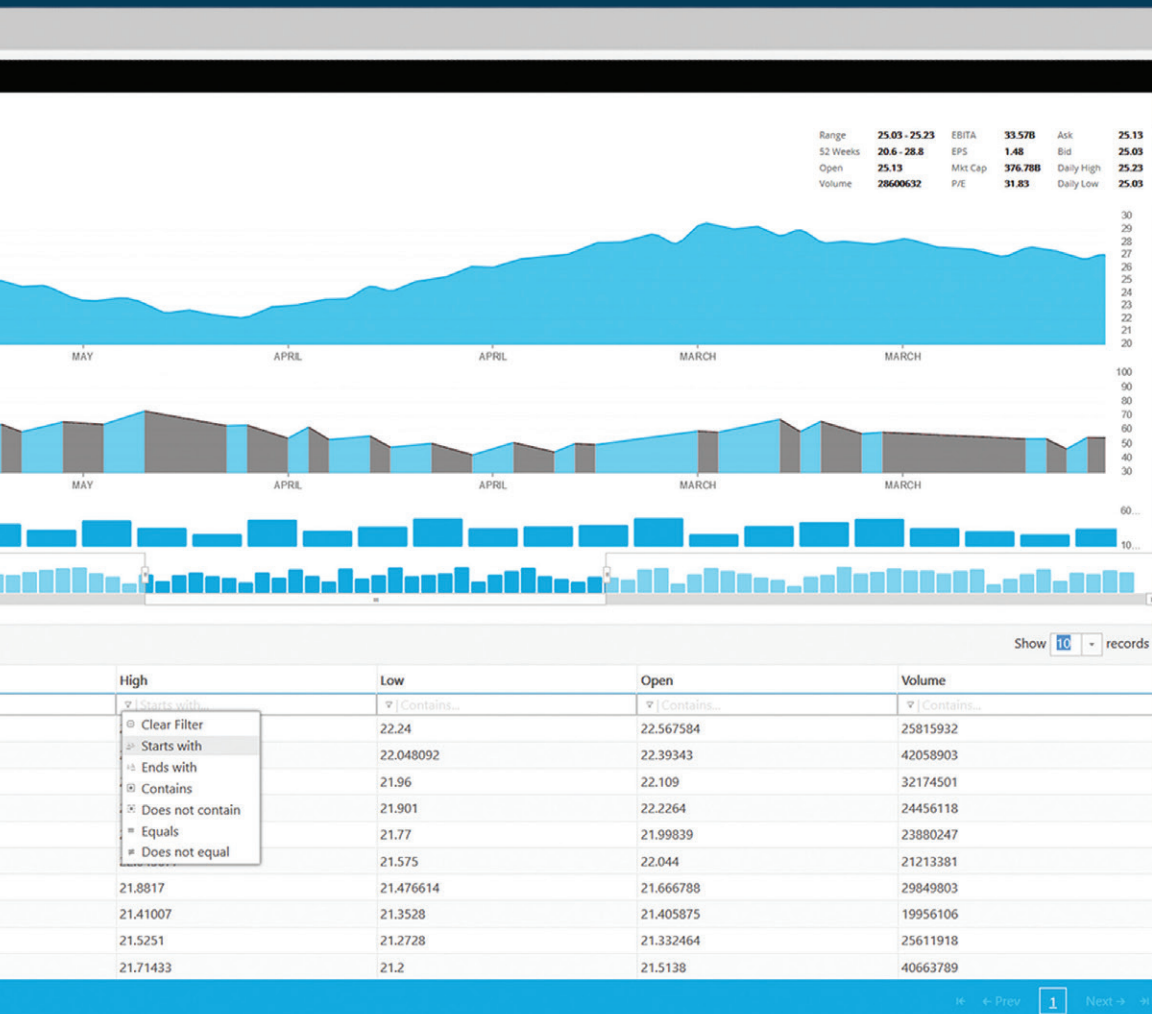
Get started today with a free trial:
Infragistics.com/Ulimate



New Release

Infragistics Ultimate 18.1

- ✓ Fastest **grids & charts** on the market for the Angular developer
- ✓ The most complete **Microsoft Excel & Spreadsheet Solution** for .NET & JavaScript
- ✓ UI controls designed to meet the demands of the toughest **financial & capital market apps**



msdn magazine

MAY 2018 VOLUME 33 NUMBER 5

General Manager Jeff Sandquist

Director Dan Fernandez

Editorial Director Jennifer Mashkowski mmeditor@microsoft.com

Site Manager Kent Sharkey

Editorial Director, Enterprise Computing Group Scott Bekker

Editor in Chief Michael Desmond

Features Editor Sharon Terdeman

Group Managing Editor Wendy Hernandez

Senior Contributing Editor Dr. James McCaffrey

Contributing Editors Dino Esposito, Frank La Vigne, Julie Lerman, Mark Michaelis, Ted Neward, David S. Platt

Vice President, Art and Brand Design Scott Shultz

Art Director Joshua Gould



President
Henry Allain

Chief Revenue Officer
Dan LaBianca

ART STAFF

Creative Director Jeffrey Langkau
Associate Creative Director Scott Rovin
Art Director Michele Singh
Art Director Chris Main
Senior Graphic Designer Alan Tao
Senior Web Designer Martin Peace

PRODUCTION STAFF

Print Production Manager Peter B. Weller
Print Production Coordinator Lee Alexander

ADVERTISING AND SALES

Chief Revenue Officer Dan LaBianca
Regional Sales Manager Christopher Kourtoglou
Advertising Sales Associate Tanya Egenolf

ONLINE/DIGITAL MEDIA

Vice President, Digital Strategy Becky Nagel
Senior Site Producer, News Kurt Mackie
Senior Site Producer Gladys Rama
Site Producer, News David Ramel
Director, Site Administration Shane Lee
Front-End Developer Anya Smolinski
Junior Front-End Developer Casey Rysavy
Office Manager & Site Assoc. James Bowling

LEAD SERVICES

Vice President, Lead Services Michele Imgrund
Senior Director, Audience Development & Data Procurement Annette Levee
Director, Audience Development & Lead Generation Marketing Irene Fincher
Director, Client Services & Webinar Production Tracy Cook
Director, Lead Generation Marketing Eric Yoshizuru
Senior Program Manager, Client Services & Webinar Production Chris Flack
Project Manager, Lead Generation Marketing Mahal Ramos

ENTERPRISE COMPUTING GROUP EVENTS

Vice President, Events Brent Sutton
Senior Director, Operations Sara Ross
Senior Director, Event Marketing Mallory Bastionell
Senior Manager, Events Danielle Potts
Coordinator, Event Marketing Michelle Cheng
Coordinator, Event Marketing Chantelle Wallace



Chief Executive Officer
Rajeev Kapur

Chief Operating Officer
Henry Allain

Chief Financial Officer
Craig Rucker

Chief Technology Officer
Erik A. Lindgren

Executive Vice President
Michael J. Valenti

Chairman of the Board
Jeffrey S. Klein

ID STATEMENT MSDN Magazine (ISSN 1528-4859) is published 13 times a year, monthly with a special issue in November by 1105 Media, Inc., 9201 Oakdale Avenue, Ste. 101, Chatsworth, CA 91311. Periodicals postage paid at Chatsworth, CA 91311-9998, and at additional mailing offices. Annual subscription rates payable in US funds are: U.S. \$35.00, International \$60.00. Annual digital subscription rates payable in U.S. funds are: U.S. \$25.00, International \$25.00. Single copies/back issues: U.S. \$10, all others \$12. Send orders with payment to: MSDN Magazine, P.O. Box 3167, Carol Stream, IL 60132, email MSDNmag@1105service.com or call (847) 763-9560. POSTMASTER: Send address changes to MSDN Magazine, P.O. Box 2166, Skokie, IL 60076. Canada Publications Mail Agreement No: 40612608. Return Undeliverable Canadian Addresses to Circulation Dept. or XPO Returns: P.O. Box 201, Richmond Hill, ON L4B 4R5, Canada.

Printed in the U.S.A. Reproductions in whole or part prohibited except by written permission. Mail requests to "Permissions Editor," c/o MSDN Magazine, 4 Venture, Suite 150, Irvine, CA 92618.

LEGAL DISCLAIMER The information in this magazine has not undergone any formal testing by 1105 Media, Inc. and is distributed without any warranty expressed or implied. Implementation or use of any information contained herein is the reader's sole responsibility. While the information has been reviewed for accuracy, there is no guarantee that the same or similar results may be achieved in all environments. Technical inaccuracies may result from printing errors and/or new developments in the industry.

CORPORATE ADDRESS 1105 Media, 9201 Oakdale Ave. Ste 101, Chatsworth, CA 91311 1105media.com

MEDIA KITS Direct your Media Kit requests to Chief Revenue Officer Dan LaBianca, 972-687-6702 (phone), 972-687-6799 (fax), dlabianca@1105media.com

REPRINTS For single article reprints (in minimum quantities of 250-500), e-prints, plaques and posters contact: PARS International
Phone: 212-221-9595
E-mail: 1105reprints@parsintl.com
Web: 1105Reprints.com

LIST RENTAL This publication's subscriber list is not available for rental. However, other lists from 1105 Media, Inc. can be rented. For more information, please contact our list manager: Jane Long, Merit Direct
Phone: 913-685-1301;
E-mail: jlong@meritdirect.com;
Web: meritdirect.com/1105

Reaching the Staff

Staff may be reached via e-mail, telephone, fax, or mail. E-mail: To e-mail any member of the staff, please use the following form: FirstInitialLastname@1105media.com
Irvine Office (weekdays, 9:00 a.m. – 5:00 p.m. PT)
Telephone 949-265-1520; Fax 949-265-1528
4 Venture, Suite 150, Irvine, CA 92618
Corporate Office (weekdays, 8:30 a.m. – 5:30 p.m. PT)
Telephone 818-814-5200; Fax 818-734-1522
9201 Oakdale Avenue, Suite 101, Chatsworth, CA 91311
The opinions expressed within the articles and other contents herein do not necessarily express those of the publisher.



LEADTOOLS®



One SDK, Multiple Platforms

LEADTOOLS toolkits will help you create desktop, web, server, and mobile applications with the greatest collection of programmer-friendly and cross-platform document, medical, and multimedia technologies.

OCR

FORMS

BARCODE

DICOM

PACS

+ MANY
MORE

Get Started Today

DOWNLOAD OUR FREE EVALUATION

LEADTOOLS.COM



Improving UWP

Andrew Whitechapel wrote this month's lead feature article, "Closing UWP-Win32 Gaps," which describes three important new capabilities added to Universal Windows Platform (UWP) apps and how developers can access them. Enabled as part of the Windows 10 version 1803 update is support in UWP apps for multi-instancing, broadened file system access, and the ability to write headless console apps in UWP. All three bring to UWP capabilities that have long been present in Win32 and .NET Framework.

"We had heard from various sources that these were important gaps in the UWP story—from internal app teams, from the MVP community and from other customers who are building or considering building UWP apps," says Whitechapel, who is a program manager in the Microsoft Windows division. "Multi-instancing was largely seen as a no-brainer."

Accessing other areas of local or remote storage required invoking a FilePicker dialog and prompting the user to choose the location—a clumsy process that breaks the UI flow.

He doesn't expect developers to scramble to add the capability to most existing UWP apps, saying that "only more complex apps with a sophisticated UX would take advantage initially," although he expects usage to increase over time as multi-instancing reveals new app opportunities. But for those apps that need it, the new support eliminates fragile and complex multi-windowing schemes as a workaround. To best support developers, Microsoft introduced two tiers of multi-instancing: a simple implementation that requires

only a manifest be added to a new or existing UWP project, and a more granular implementation that relies on redirection APIs.

Another key improvement is extended file system support. As a security measure, UWP apps by default were limited to select folders such as the Pictures or Music library. Accessing other areas of local or remote storage required invoking a FilePicker dialog and prompting the user to choose the location—a clumsy process that breaks the UI flow. Now UWP apps gain implicit access to files and folders in the current working directory and below.

"We were very cautious in pursuing broad file system access because of the obvious security and privacy concerns," Whitechapel says. "Ultimately, we felt that the restricted capability, plus the user-consent prompt—coupled with the user's ability to disable this access for any given app, or globally for all apps—was enough to mitigate these concerns."

In fact, file system access was improved as part of the effort to enable console app support in UWP—an advancement of particular import to developers and IT admins. As Whitechapel tells it, the dev team realized the challenge facing canonical console apps, like findstr and grep, that search for patterns within files.

"Such an app would be severely crippled—to the point of being almost useless—if it couldn't get broader access to the file system," he says.

Interest in console app development for UWP has been sharp. Whitechapel says that the Visual Studio Console UWP project templates have recorded more than 20,000 installs since they were released. And he says that file system access and multi-instancing have both produced "vociferous support."

So what's on tap for UWP going forward? Whitechapel points to the Windows Developer Day keynote (youtu.be/D6YAJxFsmuM) by Microsoft Corporate Vice President Kevin Gallo, where he discussed efforts to help desktop app developers modernize their code with new technologies like Modern WebView and MSIX containerization. He also singles out the Internet of Things and artificial intelligence as areas of heavy investment.

Visit us at msdn.microsoft.com/magazine. Questions, comments or suggestions for *MSDN Magazine*? Send them to the editor: mmeditor@microsoft.com.

© 2018 Microsoft Corporation. All rights reserved.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, you are not permitted to reproduce, store, or introduce into a retrieval system *MSDN Magazine* or any part of *MSDN Magazine*. If you have purchased or have otherwise properly acquired a copy of *MSDN Magazine* in paper format, you are permitted to physically transfer this paper copy in unmodified form. Otherwise, you are not permitted to transmit copies of *MSDN Magazine* (or any part of *MSDN Magazine*) in any form or by any means without the express written permission of Microsoft Corporation.

A listing of Microsoft Corporation trademarks can be found at microsoft.com/library/toolbar/3.0/trademarks/en-us.mspx. Other trademarks or trade names mentioned herein are the property of their respective owners.

MSDN Magazine is published by 1105 Media, Inc. 1105 Media, Inc. is an independent company not affiliated with Microsoft Corporation. Microsoft Corporation is solely responsible for the editorial contents of this magazine. The recommendations and technical guidelines in *MSDN Magazine* are based on specific environments and configurations. These recommendations or guidelines may not apply to dissimilar configurations. Microsoft Corporation does not make any representation or warranty, express or implied, with respect to any code or other information herein and disclaims any liability whatsoever for any use of such code or other information. *MSDN Magazine*, MSDN and Microsoft logos are used by 1105 Media, Inc. under license from owner.

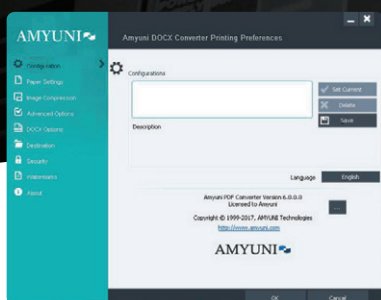


DOCXCONVERTER
For Windows

Free Demo at DOCXConverter.com

Amyuni DOCX Converter for Windows

Convert any document, including PDF documents, into DOCX format.
Enable editing of documents using Microsoft Word or other Office products.



A standalone desktop version, a server product for automated processing or an SDK for integration into third party applications.

Create

Create naturally editable DOCX documents with paragraph formatting and reflow of text

Convert

Convert images and graphics of multiple formats into DOCX shapes

OCR

Use OCR technology to convert non-editable text into real text

Extract

Extract headers and footers from source document and save them as DOCX headers and footers

Open

Open PDF documents with the integrated PDF viewer and quickly resave them to DOCX format

Configure

Configure the way the fonts are embedded into the DOCX file for optimal formatting

A virtual printer driver available for Windows 7 to Windows 10 and Windows Server 2008 to 2016

Powered by Amyuni Technologies:

Developers of the Amyuni PDF Converter and Amyuni PDF Creator products integrated into hundreds of applications and installed on millions of desktops and servers worldwide.

www.docxconverter.com

All trademarks are property of their respective owners. © Amyuni Technologies Inc. All rights reserved.



Text Sentiment Analysis

One of the truisms of the modern data-driven world is that the velocity and volume of data keeps increasing. We're seeing more data generated each day than ever before in human history. And nowhere is this rapid growth more evident than in the world of social media, where users generate content at a scale previously unimaginable. Twitter users, for example, collectively send out approximately 6,000 tweets every second, according to tracking site Internet Live Stats (internetlivestats.com/twitter-statistics). At that rate, there are about 350,000 tweets sent per minute, 500 million tweets per day, and about 200 billion tweets per year. Keeping up with this data stream to evaluate content would be impossible even for the largest teams—you just couldn't hire enough people to scan Twitter to evaluate the sentiment of its user base at any given moment.

Fortunately, the use case for analyzing *every* tweet would be an extreme edge case. There are, however, valid business motives for tracking sentiment, be it against a specific topic, search term or hashtag. While this narrows the number of tweets to analyze significantly, the sheer volume of the data to analyze still makes it impractical to analyze the sentiments of the tweets in any meaningful way.

Thankfully, analyzing the overall sentiment of text is a process that can easily be automated through sentiment analysis. Sentiment analysis is the process of computationally classifying and categorizing opinions expressed in text to determine whether the attitude expressed within demonstrates a positive, negative or neutral tone. In short, the process can be automated and distilled to a mathematical score indicating tone and subjectivity.

Setting Up an Azure Notebook

In February (msdn.com/magazine/mt829269), I covered in detail Jupyter notebooks and the environments in which they can run. While any Python 3 environment can run the code in this article, for the sake of simplicity, I'll use Azure Notebooks. Browse over to the Azure Notebooks service Web site at notebooks.azure.com and sign in with your Microsoft ID credentials. Create a new Library with the name Artificially Intelligent. Under the Library ID field enter "ArtificiallyIntelligent" and click Create. On the following page, click on New to create a new notebook. Enter a name in the Item Name textbox, choose Python 3.6 Notebook from the Item type dropdown list and click New (Figure 1).

Click on the newly created notebook and wait for the service to connect to a kernel.

Code download available at bit.ly/2pPFIMM.

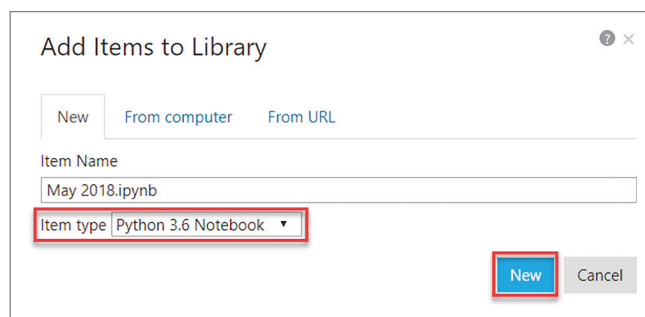


Figure 1 Creating a New Notebook with a Python 3.6 Kernel

Sentiment Analysis in Python

Once the notebook is ready, enter the following code in the empty cell and run the code in the cell.

```
from textblob import TextBlob
simple_text = TextBlob("Today is a good day for a picnic.")
print(simple_text.sentiment)
```

The results that appear will resemble the following:

```
Sentiment(polarity=0.7, subjectivity=0.6000000000000001)
```

Polarity refers to how negative or positive the tone of the input text rates from -1 to +1, with -1 being the most negative and +1 being the most positive. Subjectivity refers to how subjective the statement rates from 0 to 1 with 1 being highly subjective. With just three lines of code, I could analyze not just sentiment of a fragment of text, but also its subjectivity. How did something like sentiment analysis, once considered complicated, become so seemingly simple?

Python enjoys a thriving ecosystem, particularly in regard to machine learning and natural language processing (NLP). The code snippet above relies on the TextBlob library (textblob.readthedocs.io/en/dev). TextBlob is an open source library for processing textual data, providing a simple API for diving into common natural language processing (NLP) tasks. These tasks include sentiment analysis and much more.

In the blank cell below the results, enter the following code and execute it:

```
simple_text = TextBlob("the sky is blue.")
print(simple_text.sentiment)
```

The results state that the phrase "the sky is blue" has a polarity of 0.0 and a subjectivity of 0.1. This means that the text is neutral in tone and scores low in subjectivity. In the blank cell immediately underneath the results, enter the following code and execute the cell:

```
simple_text1 = TextBlob("I hate snowstorms.")
print(simple_text1.sentiment)

simple_text2 = TextBlob("Bacon is my favorite!")
print(simple_text2.sentiment)
```

Note that the algorithm correctly identified that the contents of `simple_text1` had a negative sentiment (-0.8) and that the statement is quite subjective (0.9). Additionally, the algorithm correctly inferred the positive sentiment of `simple_text2` (0.625) and its highly subjective nature (1.0).

However, the algorithm does have significant difficulties parsing the more subtle nuances of human language. Sarcasm, for instance, is not only hard to detect, but may throw off the results. Imagine a scenario where a restaurant extracts reviews from an online review site like Yelp and automatically publishes reviews with a positive sentiment on their Web site and social media. Enter the following code into an empty cell and execute it:

```
sample_customer_review1 =
    TextBlob("The burgers at this place will
make you ill with joy.")
print(sample_customer_review1.sentiment)

sample_customer_review2 =
    TextBlob("Whenever I want to take a sick
day, I eat here the night before and it is
always a sure fire win!")
print(sample_customer_review2.sentiment)
```

Clearly the sentiment of these two reviews are negative. Yet the algorithm seems to think otherwise, with both reviews scoring as having positive sentiment scores, 0.15 and 0.26 respectively. In this case, the restaurant would likely not want either of these reviews highlighted on any platform. NLP systems have yet to grasp a good understanding of sarcasm, although there is a lot of research currently being done in this area (thesarcasmdetector.com/about).

Connecting to Twitter

So far, I have only run small bits of text through the `TextBlob` analyzer. A more practical use of this technology is to feed it user-generated data, ideally in near-real time. Fortunately Twitter, with its approximately 327 million active users (bit.ly/2E3IYKI), provides a constant stream of text to analyze.

To connect with Twitter's API, I need to register an application with Twitter to generate the necessary credentials. In a browser, go to apps.twitter.com and, if needed, log in with your Twitter credentials. Click the Create New App button to bring up the Create an application form as shown in **Figure 2**. Enter a name, description and a Web site for the app. For the purposes

Application Management

Create an application

Application Details

Name *
MSDNTwitterSentiment
Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *
Sentiment analysis
Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *
http://www.franksworld.com
Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL
Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Developer Agreement
☒ Yes, I have read and agree to the [Twitter Developer Agreement](#).

Create your Twitter application

Figure 2 The Twitter Create Application Form

Application Management

MSDNTwitterSentiment

Test OAuth

Details Settings Keys and Access Tokens Permissions

Application Settings
Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key) tWdt

Consumer Secret (API Secret) UA3n...ISc

Access Level Read and write ([modify app permissions](#))

Owner Tableteer

Owner ID 7888762

Application Actions
Regenerate Consumer Key and Secret Change App Permissions

Your Access Token
You haven't authorized this application for your own account yet.
By creating your access token here, you will have everything you need to make API calls right away. The access token generated will be assigned your application's current permission level.

Token Actions
Create my access token

Figure 3 Twitter Application Keys and Access Tokens Screen

of this article, the Web site address does not matter, so enter a valid URL. Click the checkbox to agree to the terms of the Twitter Developer Agreement and click the Create your Twitter application button.

On the following screen, look for Consumer Key (API Key) under the Application Settings section. Click on the “manage keys and access tokens” link. On the page that follows, click the Create my access token button, as shown in **Figure 3**, to create an access token. Make note of the following four values shown on this page: Consumer Key (API Key), Consumer Secret (API Secret), Access Token and Access Token Secret.

Using Tweepy to Read Tweets

Tweepy is a Python library that simplifies the interaction between Python code and the Twitter API. More information about Tweepy can be found at docs.tweepy.org/en/v3.5.0. At this time, return to the Jupyter notebook and enter the following code to install the Tweepy API. The exclamation mark instructs Jupyter to execute a command in the shell:

```
!pip install tweepy
```

Once the code executes successfully, the response text in the cell will read: “Successfully installed tweepy-3.6.0,” although the specific version number may change. In the following cell, enter the code in **Figure 4** into the newly created empty cell and execute it.

The results that come back should look similar to the following:

```
#ElonMusk deletes own, #SpaceX and #Tesla Facebook pages after
#deletefacebook https://t.co/zKGg4ZM2pi https://t.co/d9YFboUUAJ
Sentiment(polarity=0.0, subjectivity=0.0)
RT @loislane28: Wow. did @elonmusk just delete #SpaceX and #Tesla from
Facebook? https://t.co/iN4N4zkncA
Sentiment(polarity=0.0, subjectivity=0.0)
```

Keep in mind that as the code executes a search on live Twitter data, your results will certainly vary. The formatting is a little confusing to read. Modify the for loop in the cell to the following and then re-execute the code.

```
for tweet in spacex_tweets:
    analysis = TextBlob(tweet.text)
    print('{0} | {1} | {2}'.format(tweet.text, analysis.sentiment.polarity,
    analysis.sentiment.subjectivity))
```

Adding the pipe characters to the output should make it easier to read. Also note that the sentiment property’s two fields, polarity and subjectivity, can be displayed individually.

Load Twitter Sentiment Data Into a DataFrame

The previous code created a pipe-delineated list of tweet content and sentiment scores. A more useful structure for further analysis would be a DataFrame. A DataFrame is a two-dimensional-labeled data structure. The columns may contain different value types.

Figure 4 Use Tweepy to Access the Twitter API

```
import tweepy

consumer_key = "[Insert Consumer Key value]"
consumer_secret = "[Insert Consumer Secret value]"
access_token = "[Insert Access Token value]"
access_token_secret = "[Insert Access Token Secret value]"
authentication_info = tweepy.OAuthHandler(consumer_key, consumer_secret)
authentication_info.set_access_token(access_token, access_token_secret)
twitter_api = tweepy.API(authentication_info)
spacex_tweets = twitter_api.search("#spacex")

for tweet in spacex_tweets:
    print(tweet.text)
    analysis = TextBlob(tweet.text)
    print(analysis.sentiment)
```

Similar to a spreadsheet or SQL table, DataFrames provide a familiar and simple mechanism to work with datasets.

DataFrames are part of the Pandas library. As such, you will need to import the Pandas library along with Numpy. Insert a blank cell below the current cell, enter the following code and execute:

```
import pandas as pd
import numpy as np

tweet_list = []
for tweet in spacex_tweets:
    analysis = TextBlob(tweet.text)
    tweet_list.append({'Text': tweet.text, 'Polarity': analysis.sentiment.polarity,
    'Subjectivity': analysis.sentiment.subjectivity })
tweet_df = pd.DataFrame(tweet_list)
tweet_df
```

The results now will display in an easier to read tabular format. However, that’s not all that the DataFrames library can do. Insert a blank cell below the current cell, enter the following code, and execute:

```
print ("Polarity Stats")
print ("Avg", tweet_df[["Polarity"]].mean())
print ("Max", tweet_df[["Polarity"]].max())
print ("Min", tweet_df[["Polarity"]].min())
print ("Subjectivity Stats")
print ("Avg", tweet_df[["Subjectivity"]].mean())
print ("Max", tweet_df[["Subjectivity"]].max())
print ("Min", tweet_df[["Subjectivity"]].min())
```

By loading the tweet sentiment analysis data into a DataFrame, it’s easier to run and analyze the data at scale. However, these descriptive statistics just scratch the surface of the power that DataFrames provide. For a more complete exploration of Pandas DataFrames in Python, please watch the webcast, “Data Analysis in Python with Pandas,” by Jonathan Wood at bit.ly/2urCxQX.

Wrapping Up

With the velocity and volume of data continuing to rise, businesses large and small must find ways to leverage machine learning to make sense of the data and turn it into actionable insight. Natural Language Processing, or NLP, is a class of algorithms that can analyze unstructured text and parse it into machine-readable structures, giving access to one of the key attributes of any body of text—sentiment. Not too long ago, this was out of reach of the average developer, but now the TextBlob Python library brings this technology to the Python ecosystem. While the algorithms can sometimes struggle with the subtleties and nuances of human language, they provide an excellent foundation for making sense of unstructured data.

As demonstrated in this article, the effort to analyze a given block of text for sentiment in terms of negativity or subjectivity is now trivial. Thanks to a vibrant Python ecosystem of third-party open source libraries, it’s also easy to source data from live social media sites, such as Twitter, and pull in users’ tweets in real time. Another Python library, Pandas, simplifies the process to perform advanced analytics on this data. With thoughtful analysis, businesses can monitor social media feeds and obtain awareness of what customers are saying and sharing about them. ■

FRANK LA VIGNE leads the Data & Analytics practice at Wintellect and co-hosts the *DataDriven* podcast. He blogs regularly at [FranksWorld.com](https://franksworld.com) and you can watch him on his YouTube channel, “Frank’s World TV” ([FranksWorld.TV](https://franksworld.tv)).

THANKS to the following technical expert for reviewing this article:

Andy Leonard



DevExpress Spreadsheet for WPF & WinForms

with Chart, Pivot Table and Cell Editor support

Your users already know Microsoft® Excel® and with the new capabilities we've introduced in our Spreadsheet Control, you can create solutions that utilize a "spreadsheet-first" UX for a broad range of use-case scenarios. Imagine sales entry forms that are an actual invoice or an inventory management document with built-in ordering and data entry options.

Give our Spreadsheet a try today and see how easy it is to build Windows® apps that amaze.



Free 30-day trial
devexpress.com/spreadsheet

#UseTheBest

All trademarks or registered trademarks are property of their respective owners.

Closing UWP–Win32 Gaps

Andrew Whitechapel

One of the important themes for the latest update of Windows has been to close the gaps between the Universal Windows Platform (UWP) and traditional Win32 app models. As part of this effort, Microsoft introduced three major enhancements:

- Multi-instancing
- Console UWP apps
- Broader file-system access

The features are related but largely independent. That is, you can create a multi-instanced app that's either a regular windowed app or a console app—and it may or may not need broader file-system access. Equally, you can create a regular non-console, non-multi-instance app that has broad file-system access. One restriction is that a console app must be configured to support multi-instancing.

This article discusses:

- Adding multi-instancing capabilities to UWP apps
- Creating console UWP apps
- Granting broader file-system access to UWP apps

Technologies discussed:

Universal Windows Platform, Win32

Code download available at:

bit.ly/2GtzM3T

Multi-Instancing

In Win32, Linux and other app model environments, multi-instancing has always been the default. In the UWP, in stark contrast, the default has always been single-instancing—and, in fact, multi-instancing wasn't supported at all until now.

Without multi-instancing, some apps have resorted to a multi-windowing architecture instead, which typically involves significant work—and results in complexity and fragility. You have to spend a lot of effort in managing your windows, instead of focusing on your domain requirements. Single-process multi-windowing also suffers from reliability concerns: If the single instance crashes, it brings down all its windows; this isn't true for multi-instancing, where each instance runs as a separate process.

In the single-instance model, the user can activate an app in a number of ways: via a tile-tap in Start; via a URL or protocol activation; by double-clicking a file with an extension that's registered to the app; and so on. The first activation (of any kind) launches the app. After that, any subsequent activations simply call into the running instance of the app, which the app can handle by overriding the `OnActivated` method.

The new multi-instancing feature enables UWP apps to work like Win32 apps: If an instance of an app is running and a subsequent activation request is received, the platform won't call in to activate the existing instance. Instead, it will create a new instance in a separate process.

Because this feature is largely driven by Win32 parity, it's initially supported only on Desktop and IoT. Two levels of support for multi-instancing were introduced:

- **Multi-Instance UWP App:** This is for the simple case, where the app just wants to declare that it should be multi-instanced.
- **Multi-Instance Redirection UWP App:** This is for the complex case, where the app wants to be multi-instanced, but it also wants to have a say in exactly how each instance is activated.

For both cases, a Visual Studio project template is provided, as shown in **Figure 1**.

For the simple case, the project template generates code that's almost identical to the Blank Application template code. The only difference is in the use of the `SupportsMultipleInstances` attribute in the app manifest. There are two additional declarations: The first is for the desktop4 and iot2 XML namespaces at the top of the manifest:

```
xmlns:desktop4="http://schemas.microsoft.com/appx/manifest/desktop/windows10/4"
xmlns:iot2="http://schemas.microsoft.com/appx/manifest/iot/windows10/2"
IgnorableNamespaces="uap mp desktop4 iot2">
```

The second adds the attribute on the `<Application>` element:

```
<Application
  Id="App8" Executable="$targetnametoken$.exe" EntryPoint="App8.App"
  desktop4:SupportsMultipleInstances="true"
  iot2:SupportsMultipleInstances="true">
```

If the platform doesn't indicate a preferred instance, you go ahead and compose a key.

If you're updating existing app code rather than generating a new app, you can simply add these entries to the manifest manually. Once you've done this, you can build your app and launch multiple instances. With this manifest entry, every time your app is activated—whether from a tile-tap, or any other activation contract the app supports, such as file-association or protocol-launch—each activation will result in a separate instance. It's as simple as that.

Multi-Instance Redirection

For most apps, all you need to do is add the manifest entry, but for an app that wants a finer degree of control over its instance activations, you can use the second template. This adds the exact same manifest entries, and also adds an additional file (Program.cs for C# apps, or Program.cpp for C++) that contains a standard Main

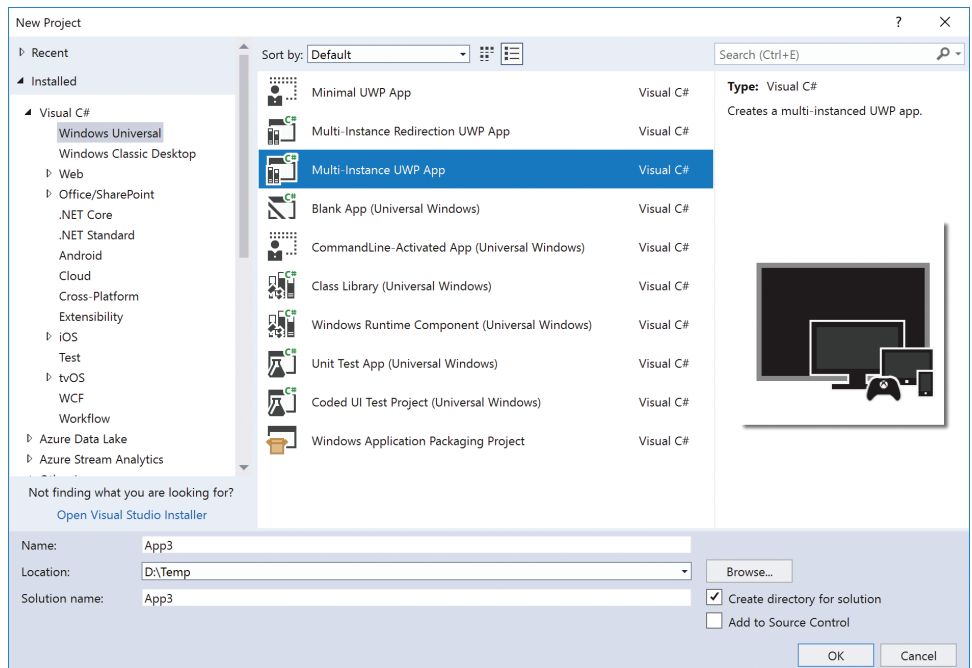


Figure 1 New Project Templates for Multi-Instanced Apps

function, as shown in **Figure 2**. This uses a new `AppInstance` class introduced in this release.

The first thing the function does is grab this instance's activation arguments. The app would likely use the information held in these arguments as part of its redirection logic.

In some scenarios, the platform might indicate a recommended instance. If so, you can redirect this activation to that instance instead, if you wish, by using the `AppInstance.RedirectActivationTo()` method. In other words, the app can choose to allow this activation request to be redirected to an existing instance instead. If it does redirect, then the target instance is activated and its `OnActivated` method is invoked—and this new instance is terminated.

If the platform doesn't indicate a preferred instance, you go ahead and compose a key. The example code composes a key from a

Figure 2 Standard Main Function for Multi-Instance Redirection

```
static void Main(string[] args)
{
    IActivatedEventArgs activatedArgs = AppInstance.GetActivatedEventArgs();

    if (AppInstance.RecommendedInstance != null)
    {
        AppInstance.RecommendedInstance.RedirectActivationTo();
    }
    else
    {
        uint number = CryptographicBuffer.GenerateRandomNumber();
        string key = (number % 2 == 0) ? "even" : "odd";
        var instance = AppInstance.FindOrRegisterInstanceForKey(key);
        if (instance.IsCurrentInstance)
        {
            global::Windows.UI.Xaml.Application.Start((p) => new App());
        }
        else
        {
            instance.RedirectActivationTo();
        }
    }
}
```

random number, but you'd normally replace this code and compose a key based on app-defined logic. Typically this would be based on the activation arguments retrieved earlier. For example, if the activation arguments were of type `FileActivatedEventArgs`, the app might use the specified file name as part of the key. Once you've composed the key, you pass it to the `FindOrRegisterInstanceForKey` method, which returns an `AppInstance` object that represents an instance of this app. To determine which instance to return, the method does two things:

- It searches for an existing instance of the app that has already registered this key.
- If no existing instance has already registered this key, it registers this current instance with this key.

If you've successfully registered this instance, you can now just go ahead and do the normal app initialization. For a XAML app, this means calling `Application.Start` with a new instance of the `App` class. If some other instance has already registered this key, you can now redirect this activation to that instance instead, and allow this instance to terminate. For example, consider an app that edits files. If the user has `Foo.doc` open for edits in the app, and then attempts to open `Foo.doc` again, the app might choose to redirect the second activation to the instance that already has `Foo.doc` open—and prevent the user from opening the same file in multiple instances. The logic for deciding whether or not to redirect, and which instance to select as the redirection target, is entirely app-defined.

For XAML apps, the `Main` method is normally auto-generated and hidden from the developer. This behavior is suppressed in the “multi-instance with redirection” template. If you're updating an existing app, you can suppress the default `Main` by adding `DISABLE_XAML_GENERATED_MAIN` to the list of conditional compilation symbols in the build properties for the app. In some app types—for example, a C++ DirectX app—the `Main` function isn't hidden. Apart from that, a DirectX app's use of the new APIs follows the same patterns as in the XAML example.

Note that an app can only use the `GetActivatedEventArgs` and `RedirectActivationTo` methods during `Main`; if these are called anywhere else, they will fail. This is because if you want to take part in activation redirection, you need to do this extremely early in the life of the app process, and certainly before any windows are created.

On the other hand, you can use the remaining `AppInstance` methods and properties at any time. In particular, you can use `FindOrRegisterInstanceForKey` to update the key for the current instance whenever you need to. For example, if your key was based on a file name, and you later close this file, you'd update your key registration at that time. You can also use the `Unregister` method to unregister completely if for some reason you no longer want this particular instance to take part in activation redirection. Also, at any time, you can use the `AppInstance.GetInstances` method to get a list of all registered instances of your app, including their keys, so that you can reason about their state.

Additional Considerations

Multi-instancing is a major enhancement, and the initial release covers only the major scenarios. Specifically, support is included for multi-instancing a foreground app, console apps, and most

out-of-process background tasks including app services. However, there's no support in this release for `ApplicationTrigger` tasks or any in-proc background tasks.

During development, Microsoft spent considerable time testing a broad range of existing Store apps to see how they'd perform when multi-instanced. From this, Microsoft learned that apps fall into three broad categories:

- Apps that have no reason to be multi-instanced. These apps just won't opt in to the feature.
- Apps that want to be multi-instanced, and continue to function correctly without any code changes. These apps can simply opt in to multi-instancing and call it done.
- Apps that want to be multi-instanced, but need to do work to allow for the differences in execution model.

For XAML apps,
the `Main` method is normally
auto-generated and hidden
from the developer.

The common issue with apps in the third category is that they're using some central resource—perhaps a cache, or a database, or other file—and when single-instanced they've been safely assuming that the app has exclusive access to this resource. Once they opt in to multi-instancing, there may be multiple instances attempting to access the resource. In this scenario, the app needs to do work to synchronize access, locking reads and writes, and so on—in other words, all the usual synchronization issues that traditional Win32 apps need to consider.

As an obvious example, consider the use of the app's local storage. This is an example of a resource where access is constrained on a package basis, not a process basis—and of course all instances of an app share the same package. Although each instance of the app is running as a separate process, they will all use the same local storage and settings, as represented by the `ApplicationData.Current` API. If you're performing data access operations in local storage, you should consider how to protect against clashes. One option is to use instance-unique files, where one instance's operations can't conflict with any other's. Alternatively, if you want to use a common file across multiple instances, you should lock and unlock access to the file appropriately. You can use standard mechanisms such as a named `Mutex` for this.

Console UWP Apps

Another obvious gap in the UWP landscape is the ability to create a headless console app. In Win32 and other environments, you can create a command-line tool that uses the console window for input and output. So, we added this support also. Again, there's a new Visual Studio project template, and as with multi-instanced apps, this generates additional manifest entries. This feature is also restricted to Desktop and IoT—not least because only those SKUs actually

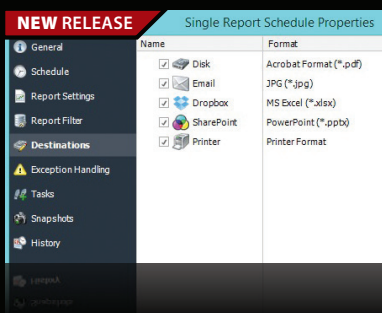


Help & Manual Professional | from \$586.04



Help and documentation for .NET and mobile applications.

- Powerful features in an easy, accessible and intuitive user interface
- As easy to use as a word processor, but with all the power of a true WYSIWYG XML editor
- Single source, multi-channel publishing with conditional and customized output features
- Output to responsive HTML, CHM, PDF, MS Word, ePub, Kindle or print
- Styles and Templates give you full design control



PBRS (Power BI Reports Scheduler) | from \$8,132.20

christianstevenson

Data Driven Distribution for Power BI Reports & Dashboards.

- A comprehensive set of job (schedule) types gives you the power to automate delivery in Power BI
- Automate report delivery & send reports to printer, fax, folder, FTP, DropBox, SharePoint & email
- Contains powerful system event triggered, data-driven and business process workflow functions
- Respond instantly by firing off reports and automation scripts when an event occurs



Oxygen XML Editor Professional | from \$660.48



The complete XML editing solution, both for XML developers and content authors

- Produce output in PDF, ePub, HTML, and many other formats from a single source
- Ready-to-use support for DITA, DocBook, XHTML, and TEI frameworks
- Extend the built-in XML publishing frameworks, or even create your own frameworks
- Interact with the majority of XML databases, content management systems, and WebDAV
- Make sure your XML documents are "well-formed" and valid, using as-you-type validation



DevExpress DXperience 17.2 | from \$1,439.99



The complete range of DevExpress .NET controls and libraries for all major Microsoft platforms.

- WinForms - New TreeMap control, Chart series types and Unbound Data Source
- WPF - New Wizard control and Data Grid scrollbar annotations
- ASP.NET - New Vertical Grid control, additional Themes, Rich Editor Spell Checking and more
- Windows 10 Apps - New Hamburger Sub Menus, Splash Screen and Context Toolbar controls
- CodeRush - New debug visualizer expression map and code analysis diagnostics

Figure 3 Additional Manifest Entries for a Console App

```
<Application Id="App"
  Executable="$targetname$.exe"
  EntryPoint="App9.App"
  desktop4:Subsystem="console"
  desktop4:SupportsMultipleInstances="true"
  iot2:Subsystem="console"
  iot2:SupportsMultipleInstances="true">
...
<Extensions>
  <uap5:Extension
    Category="windows.appExecutionAlias"
    Executable="App9.exe"
    EntryPoint="App9.App">
    <uap5:AppExecutionAlias
      desktop4:Subsystem="console"
      iot2:Subsystem="console">
      <uap5:ExecutionAlias Alias="App9.exe"/>
    </uap5:AppExecutionAlias>
  </uap5:Extension>
</Extensions>
</Application>
```

Figure 4 Template-Generated Code for a Console App Main Function

```
int __cdecl main()
{
    // You can get parsed command-line arguments from the CRT globals.
    wprintf(L"Parsed command-line arguments:\n");
    for (int i = 0; i < __argc; i++)
    {
        wprintf(L"  argv[%d] = %S\n", i, __argv[i]);
    }

    wprintf(L"Press Enter to continue:");
    getchar();
}
```

have console windows right now. The same XML namespaces are declared. The `<Application>` element includes both `SupportsMultipleInstances` and `Subsystem` attributes, with `Subsystem` set to “console.” Console apps must be multi-instanced—this is the expected model for apps moving from traditional Win32 console apps. In addition, the app includes an `AppExecutionAlias`—and this also has the new `Subsystem` attribute, as shown in **Figure 3**.

You can change the `Alias` value to something appropriate for your app. Again, as with multi-instancing, the code-generation includes a `Program.cs` or `Program.cpp` file. The generated code provides an example of how you could implement the required main function, as shown in the C++ example in **Figure 4**. You can replace all the code inside `main` with your own custom code.

Once you’ve built and deployed the app, you can execute it from a regular command prompt, PowerShell window, or Windows-R, as shown in **Figure 5**. Note that because the app uses the console window, it’s not expected to create any other windows—and indeed, this is not supported. Instead, the app can now use all the `System.Console` APIs, plus many traditional Win32 APIs that have now been added to the approved list specifically to support console apps.

With this feature, you can finally build command-line console apps that take advantage

of the benefits of the UWP, including APPX packaging, Store publication, easy updates and so on.

Broader File-System Access

Until now, a UWP app has only been able to access certain specific folders, such as the Pictures library and the Music library—and then only if the app declares these as capabilities in its manifest. Beyond that, the app could get access to anywhere else in the file system by raising a `FilePicker` dialog and prompting the user to choose a location, which grants the app permissions.

Now, the third major feature added for Win32 parity increases the level of file-system access for UWP apps. This was done in two ways by including:

- Implicit access to the current working directory.
- Broad file-system access gated by a restricted capability.

Any UWP app (either a regular windowed app or a console app) that declares an `AppExecutionAlias` is now granted implicit access to the files and folders in the current working directory and downward, when it’s activated from a command line. The current working directory is from whatever file-system location the user chooses to execute your `AppExecutionAlias`. This was debated for a long time, as the UWP model has always been very cautious about granting file-system access to apps. On balance, it was decided that the user choosing to execute the app from a particular location is equivalent to the user choosing a location in a `FilePicker` dialog, in terms of granting permissions.

It’s important to note that the app has exactly the same file permissions as the user who’s running the app—so there might still be files or folders the app can’t access, because the user can’t access them, either. For example, if the user can’t see a hidden file when they execute a `dir` command, the app also won’t be able to see that hidden file.

To take advantage of this feature, you can code the `OnActivated` override to look for `CommandLineActivatedEventArgs`. This will include the `CurrentDirectoryPath`, which in this case will be the file-system location from which the user executed your `AppExecutionAlias`. **Figure 6** shows an example; here, the app extracts the current directory and passes it on to the `MainPage`.

You could then code your `MainPage` `OnNavigatedTo` to override to retrieve this path from the incoming `NavigationEventArgs`, as shown in **Figure 7**. In this example, the app is initializing a `StorageFolder` from this path, and then building a `TreeView` control for the files and folders from here downward.

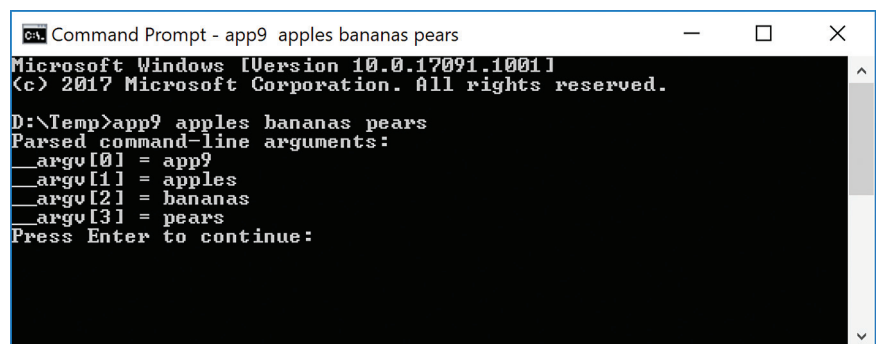


Figure 5 Executing a Console UWP App from the Command Line

File Format APIs



Open, Create, Convert, Print and Save
files from your applications!

Try risk free – 30 day trial

Download a Free Trial at
<https://downloads.aspose.com>



Aspose.Words

Create, edit, convert and print Word documents (DOC, DOCX, RTF, etc.) in your .NET, Java and Android applications.



Aspose.Cells

Develop high performance .NET, Java and Android applications to create, edit and convert Excel worksheets (XLS, XLSX, ODS, etc.).



Aspose.Pdf

Manipulate PDF file formats (PDF, PDF/A, XPS, etc.) using our native APIs for .NET, Java and Android platforms.



Aspose.Slides

Create, edit and convert PowerPoint presentations (PPT, PPTX, ODP, etc.) in your .NET, Java and Android applications.



Aspose.Email

Create, edit and convert Outlook Email file formats (MSG, PST, EML, etc.) and popular network protocols.



Aspose.BarCode

Generate or recognize barcodes (Code128, PDF417, Postnet, etc.) using our native APIs for .NET and Java.

▸ Aspose.Imaging ▸ Aspose.Tasks ▸ Aspose.OCR ▸ Aspose.Diagram ▸ Aspose.Note ▸ Aspose.HTML

Americas: +1 903 306 1676

EMEA: +44 141 628 8900
sales@asposeptyltd.com

Oceania: +61 2 8006 6987

Figure 6 Overriding OnActivated for Command-Line Activation

```
protected override void OnActivated(IActivatedEventArgs args)
{
    switch (args.Kind)
    {
        case ActivationKind.CommandLineLaunch:
            CommandLineActivatedEventArgs cmdLineArgs =
                args as CommandLineActivatedEventArgs;
            CommandLineActivationOperation operation = cmdLineArgs.Operation;
            string activationPath = operation.CurrentDirectoryPath;

            Frame rootFrame = Window.Current.Content as Frame;
            if (rootFrame == null)
            {
                rootFrame = new Frame();
                Window.Current.Content = rootFrame;
            }
            rootFrame.Navigate(typeof(MainPage), activationPath);
            Window.Current.Activate();
            break;
    }
}
```

Figure 7 Building a File-System Tree from the Current Working Directory

```
protected async override void OnNavigatedTo(NavigationEventArgs e)
{
    string activationPath = e.Parameter as string;
    argumentsText.Text = activationPath;
    fileTreeView.RootNodes.Clear();

    try
    {
        StorageFolder folder =
            await StorageFolder.GetFolderFromPathAsync(activationPath);
        if (folder != null)
        {
            TreeViewNode rootNode = new TreeViewNode() { Content = folder.Name };
            IReadOnlyList<StorageFolder> folders = await folder.GetFoldersAsync();
            GetDirectories(folders, rootNode);
            fileTreeView.RootNodes.Add(rootNode);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
}
```

New Capability

The second way that more file-system access is being provided is via a new restricted capability. To use this, you must declare the `restrictedcapabilities` XML namespace at the top of your app manifest, and include `broadFileSystemAccess` in your `<Capabilities>` list:

```
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
IgnorableNamespaces="uap mp uap5 rescap">
...
<Capabilities>
  <rescap:Capability Name="broadFileSystemAccess" />
</Capabilities>
```

If you declare any restricted capability, this triggers additional scrutiny at the time you submit your package to the Store for publication. If the app is granted this capability, it will have the same access to the file-system as the user running the app. Not just from the current working directory but everywhere that the user has access. You don't need an `AppExecutionAlias` if you have this capability. Because this is such a powerful feature, Microsoft will grant the capability only if the app developer provides compelling reasons for the request, a description of how this will be used, and an explanation of how this benefits the user.

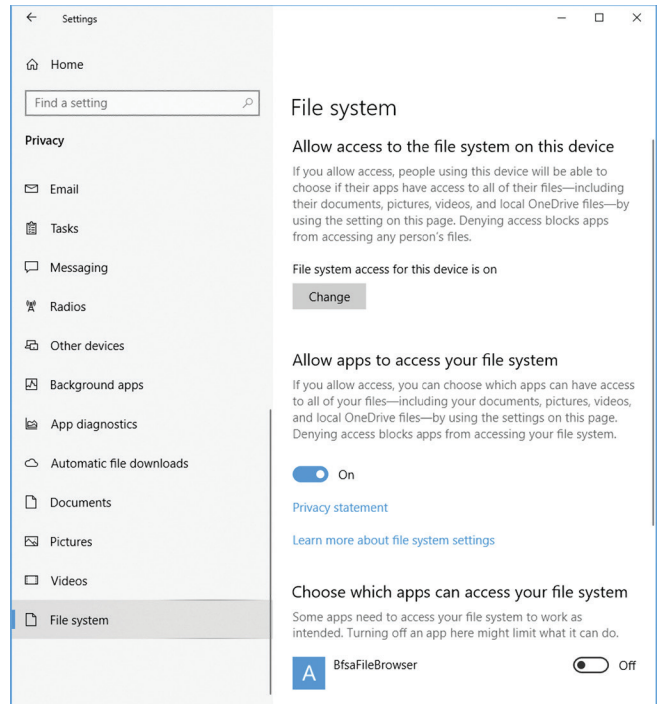


Figure 8 New File System Page in Settings

If you declare the `broadFileSystemAccess` capability, you don't need to declare any of the more narrowly scoped file-system capabilities (`Documents`, `Pictures` or `Videos`); indeed, an app must not declare both `broadFileSystemAccess` and any of the other three file-system capabilities.

Even after the app has been granted the capability, there's also a runtime check, because this constitutes a privacy concern for the user. Just like other privacy issues, the app will trigger a user-consent prompt on first use. If the user chooses to deny permission, the app must be resilient to this. The user can also change her mind at any time, by going to the relevant File system page under the Privacy list in Settings, as shown in **Figure 8**.

Note that to take advantage of both the current working directory access and the `broadFileSystemAccess` permissions, your code must use the WinRT `Windows.Storage` APIs for file handling.

Wrapping Up

One of the long-term strategies with the UWP is to close the gaps with earlier app technologies—especially Win32—so that the UWP is a viable option for more and more different app types over time. With the introduction of support for true multi-instancing, console UWP apps, and broader file-system access, three more large steps have been taken on this journey. Sample code is available at bit.ly/2GtzM3T, and you'll find the Visual Studio project templates at bit.ly/2HApmmi and bit.ly/2FEIAXu. ■

ANDREW WHITECHAPEL is a program manager in the Microsoft Windows division, responsible for the app activation workflow for the Universal Windows Platform.

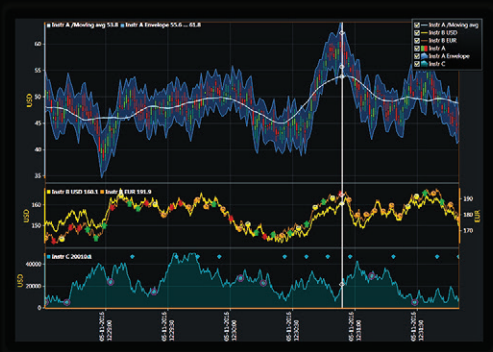
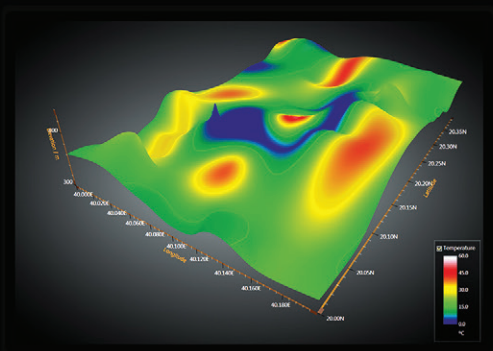
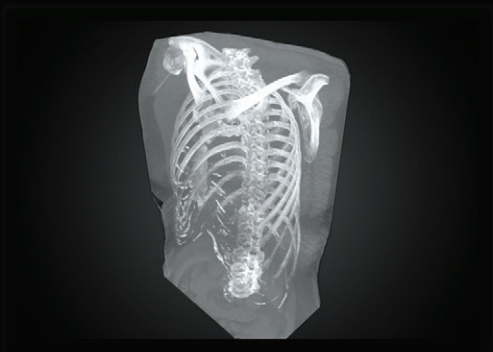
THANKS to the following technical experts for reviewing this article: Jason Holmes, Tim Kurtzman, Anis Mohammed Khaja Mohideen



[WPF]
[Windows Forms]
[Free Gauges]
[Data Visualization]
[Volume Rendering]
[3D / 2D Charts] [Maps]

LightningChart[®]

The fastest and most advanced
charting components



Create **eye-catching** and
powerful charting applications
for engineering, science
and trading

- DirectX GPU-accelerated
- Optimized for real-time monitoring
- Supports gigantic datasets
- Full mouse-interaction
- Outstanding technical support
- Hundreds of code examples

NEW

- Now with Volume Rendering extension
- Flexible licensing options

Get free trial at
LightningChart.com/ms



Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

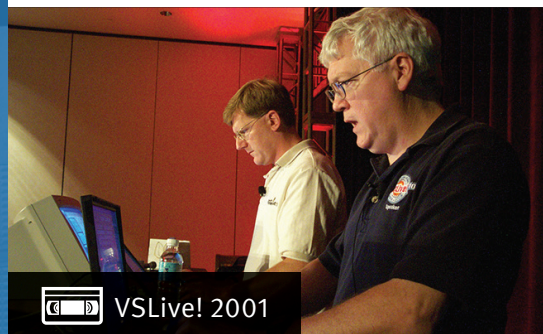
June 10-14, 2018
Hyatt Regency Cambridge

Boston



Developing Perspective

Visual Studio Live! (VSLive!™) returns to Boston, June 10 – 14, 2018, with 5 days of practical, unbiased, Developer training, including NEW intense hands-on labs. Join us as we dig into the latest features of **Visual Studio 2017, ASP.NET Core, Angular, Xamarin, UWP** and more. Code with industry experts AND Microsoft insiders while developing perspective on the Microsoft platform.



VSLive! 2001



VSLive! 2017

BACK BY POPULAR DEMAND: Pre-Con, Hands-On Labs!

Sunday, June 10
Choose from:

- › ASP.NET Core 2 and EF Core 2
- › Xamarin and Xamarin.Forms



vslive.com/bostonmsdn

**Register by May 11
and Save \$200!**

Register to code with us today!

Use promo code VSLB01

SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



AGENDA AT-A-GLANCE

Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

Boston

ALM / DevOps	Cloud Computing	Database and Analytics	Native Client	Software Practices	Visual Studio / .NET Framework	Web Client	Web Server
--------------	-----------------	------------------------	---------------	--------------------	--------------------------------	------------	------------

Pre-Conference Full Day Hands-On Labs: Sunday, June 10, 2018 <i>(Separate entry fee required)</i>					
START TIME	END TIME	Post-Conference Workshop Registration - Coffee and Morning Pastries			
7:00 AM	8:00 AM	HOL01 Full Day Hands-On Lab: Develop an ASP.NET Core 2 and EF Core 2 App in a Day - <i>Philip Japikse</i>		HOL02 Full Day Hands-On Lab: From 0-60 in a Day with Xa marin and Xamarin.Forms - <i>Roy Cornelissen & Marcel de Vries</i>	
START TIME	END TIME	Pre-Conference Workshops: Monday, June 11, 2018 <i>(Separate entry fee required)</i>			
8:00 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Morning Pastries			
9:00 AM	6:00 PM	M01 Workshop: DevOps, You Keep Saying That Word - <i>Brian Randell</i>	M02 Workshop: SQL Server for Developers - <i>Andrew Brust and Leonard Lobel</i>		M03 Workshop: Distributed Cross-Platform Application Architecture - <i>Rockford Lhotka and Jason Bock</i>
6:45 PM	9:00 PM	Dine-A-Round			
START TIME	END TIME	Day 1: Tuesday, June 12, 2018			
7:00 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	T01 Angular 101 - <i>Deborah Kurata</i>	T02 A Developer's Introduction to XR: Virtual, Augmented & Mixed Reality - <i>Nick Landry</i>	T03 SQL Server 2017 - Intelligence Built-in - <i>Scott Klein</i>	T04 Getting to the Core of the .NET Standard - <i>Adam Tuliper</i>
9:30 AM	10:45 AM	T05 ASP.NET Core 2 For Mere Mortals - <i>Philip Japikse</i>	T06 Lessons Learned from Real-World HoloLens & Mixed Reality Projects - <i>Nick Landry</i>	T07 Bots and AI with Azure - <i>Scott Klein</i>	T08 To Be Announced
11:00 AM	12:00 PM	KEYNOTE: To Be Announced			
12:00 PM	1:00 PM	Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors			
1:30 PM	2:45 PM	T09 Migrating to ASP.NET Core - A True Story - <i>Adam Tuliper</i>	T10 Re-imagining an App's User Experience: A Real-world Case Study - <i>Billy Hollis</i>	T11 Introduction to Azure Cosmos DB - <i>Leonard Lobel</i>	T12 What's New in C#7 - <i>Jason Bock</i>
3:00 PM	4:15 PM	T13 Angular Component Communication - <i>Deborah Kurata</i>	T14 There is No App - <i>Roy Cornelissen</i>	T15 SQL Server Security Features for Developers - <i>Leonard Lobel</i>	T16 Concurrent Programming in .NET - <i>Jason Bock</i>
4:15 PM	5:30 PM	Welcome Reception			
START TIME	END TIME	Day 2: Wednesday, June 13, 2018			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	W01 An Introduction to TypeScript - <i>Jason Bock</i>	W02 Strategies and Decisions for Developing Desktop Business Applications - <i>Billy Hollis</i>	W03 Applying ML to Software Development - <i>Vishwas Lele</i>	W04 Architecting Systems for Continuous Delivery - <i>Marcel de Vries</i>
9:30 AM	10:45 AM	W05 To Be Announced	W06 Cross-Platform App Dev with Xamarin and CSLA .NET - <i>Rockford Lhotka</i>	W07 Predicting the Future Using Azure Machine Learning - <i>Eric D. Boyd</i>	W08 Database DevOps - <i>Brian Randell</i>
11:00 AM	12:00 PM	GENERAL SESSION: .NET Everywhere and for Everyone <i>- James Montemagno, Principal Program Manager - Mobile Developer Tools, Microsoft</i>			
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch			
1:00 PM	1:30 PM	Dessert Break - Visit Exhibitors - Exhibitor Raffle @ 1:15pm (Must be present to win)			
1:30 PM	2:45 PM	W09 User Authentication for ASP.NET Core MVC Applications - <i>Brock Allen</i>	W10 Xamarin: The Future of App Development - <i>James Montemagno</i>	W11 Analytics and AI with Azure Databricks - <i>Andrew Brust</i>	W12 Using Feature Toggles to Help Us Separate Releases from Deployments - <i>Marcel de Vries</i>
3:00 PM	4:15 PM	W13 Securing Web APIs in ASP.NET Core - <i>Brock Allen</i>	W14 Programming with the Model-View-ViewModel Pattern - <i>Miguel Castro</i>	W15 Power BI: What Have You Done for Me Lately? - <i>Andrew Brust</i>	W16 Azure DevOps with VSTS, Docker, and K8 - <i>Brian Randell</i>
4:30 PM	5:45 PM	W17 Multi-targeting the World - <i>Oren Novotny</i>	W18 Cognitive Services in Xamarin Applications - <i>Veronika Kolesnikova & Willy Ci</i>	W19 Azure in the Enterprise - <i>Mike Benkovich</i>	W20 Visualizing the Backlog with User Story Mapping - <i>Philip Japikse</i>
6:15 PM	8:30 PM	VSLive!'s Boston By Land and Sea Tour			
START TIME	END TIME	Day 3: Thursday, June 14, 2018			
7:30 AM	8:00 AM	Registration - Coffee and Morning Pastries			
8:00 AM	9:15 AM	TH01 Tools for Modern Web Development - <i>Ben Hoelting</i>	TH02 Writing Testable Code and Resolving Dependencies - DI Kills Two Birds with One Stone - <i>Miguel Castro</i>	TH03 Containers Demystified - <i>Robert Green</i>	TH04 Busy .NET Developer's Guide to Python - <i>Ted Neward</i>
9:30 AM	10:45 AM	TH05 JavaScript Patterns for the C# Developer - <i>Ben Hoelting</i>	TH06 Netstandard: Reuse Your Code on Windows, Linux, Mac, and Mobile - <i>Rockford Lhotka</i>	TH07 Microservices with ACS (Managed Kubernetes) - <i>Vishwas Lele</i>	TH08 Signing Your Code the Easy Way - <i>Oren Novotny</i>
11:00 AM	12:15 PM	TH09 ASP Core Middleware & More - <i>Miguel Castro</i>	TH10 The Role of an Architect - <i>Ted Neward</i>	TH11 Go Serverless with Azure Functions - <i>Eric D. Boyd</i>	TH12 Get Started with Git - <i>Robert Green</i>
12:15 PM	1:15 PM	Lunch			
1:15 PM	2:30 PM	TH13 Enhancing Web Pages with VueJS: When You Don't Need a Full SPA - <i>Shawn Wildermuth</i>	TH14 Unit Testing & Test-Driven Development (TDD) for Mere Mortals - <i>Benjamin Day</i>	TH15 Computer, Make It So! - <i>Veronika Kolesnikova & Willy Ci</i>	TH16 C# 7, Roslyn and You - <i>Jim Wooley</i>
2:45 PM	4:00 PM	TH17 Versioning APIs with ASP.NET Core - <i>Shawn Wildermuth</i>	TH18 Coaching Skills for Scrum Masters & The Self-Organizing Team - <i>Benjamin Dav</i>	TH19 Compute Options in Azure - <i>Mike Benkovich</i>	TH20 Improving Code Quality with Roslyn Code Analyzers - <i>Jim Wooley</i>

Speakers and sessions subject to change

CONNECT WITH US



twitter.com/vslive – @VSLive



facebook.com – Search "VSLive"



linkedin.com – Join the "Visual Studio Live" group!

vslive.com/bostonmsdn

Building Connected Apps with UWP and Project Rome

Tony Champion

In today's world, building a successful app means moving beyond a single device. Users want apps that span all their devices and even connect with other users. Providing this type of experience can be a daunting challenge, to say the least. To help address this growing need in the ecosystem, Microsoft introduced Project Rome. Project Rome aims to create a more personal OS that spans apps, devices and users. While Project Rome has SDKs available for most major platforms, in this article I'm going to explore using Project Rome to create a team messaging Universal Windows Platform (UWP) app.

Say Hello to Project Rome

Project Rome is an initiative to help you drive user engagement across apps and devices. It's a collection of APIs that are part of

Microsoft Graph and can be categorized into two areas: continue now and continue later.

The Remote System APIs enable an app to break beyond the boundaries of the user's current device, enabling a continue-now experience. Whether allowing the user to use two devices for a single experience, as with a companion or remote-control app, or allowing multiple users to connect and share a single experience, these APIs provide an expanded view of the user's current engagement. The team messaging app built in this article will create a shared user experience.

The other half of Project Rome, the Activities APIs, focuses on continuing the user's experience at a later time. These APIs allow you to record and retrieve specific actions within your app that the user can continue from any device. Though I don't discuss them in this article, these are definitely worth looking into.

This article discusses:

- The Project Rome RemoteSystem API
- Building remote sessions
- Enabling communication between devices

Technologies discussed:

Project Rome, Universal Windows Platform, Visual Studio 2017, Visual C#

Code download available at:

bit.ly/2FWtCc5

Getting Started

Before diving into building the app, I first need to set up my environment. While the first version of Project Rome has been out for a little while, some of the features used in this article were just released during the recent Fall Creators Update. Therefore, your machine must be running build number 16299 or greater. At this point, this release is available in the slow ring for updates and most machines should be updated correctly.

Running apps with the Remote System APIs with other users requires that shared experiences are enabled on the machine. This

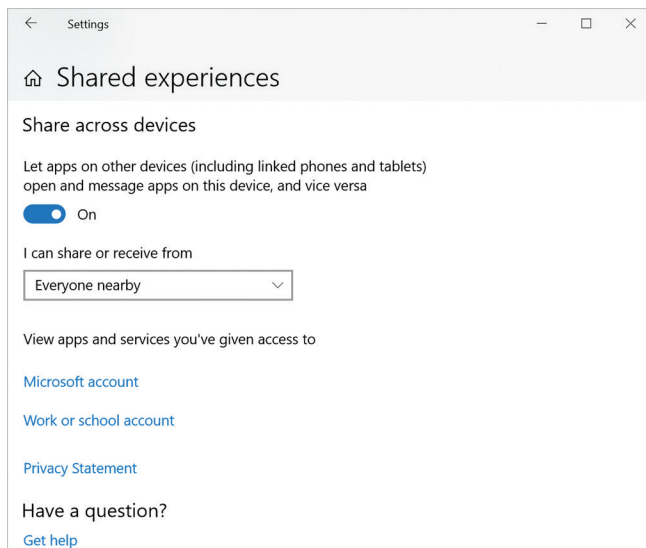


Figure 1 Enabling Shared Experiences

can be done in the system settings, in Settings | System | Shared experiences. For the team messaging scenario, you need to enable different users to communicate with your device, which means you need to make sure shared experiences is enabled and that you can share or receive from “Everyone nearby,” as shown in **Figure 1**.

The final requirement is that your device be discoverable by having some level of connectivity. The Remote System APIs will discover other machines on the same network, as well as those nearby by using Bluetooth. Bluetooth can be enabled in the “Bluetooth and other device settings” page in your system settings.

With the machine set up, let’s begin by creating a new Visual C# app using the Blank App (Universal Windows) template in Visual Studio 2017. Call the app “TeamMessenger.” As mentioned earlier, this project requires the Fall Creators Update, so set the target and minimum versions of the app to “Build 16299” or greater, as shown in **Figure 2**. This will prevent the app from supporting older versions of Windows 10, but it’s necessary for some of the features touched on in this article.

Note that if you don’t have the Fall Creators update SDKs on your device, the easiest way to obtain them is to update Visual Studio 2017 to the latest release.

The Project Rome APIs are part of the Windows 10 SDK, which means there are no additional SDKs to download or NuGet packages to install in order to build this app. There are, however, a few capabilities that must be added to the app in order for the remote session APIs to work correctly. This can be done by opening the package.appxmanifest file and selecting the Capabilities tab. In the list of available capabilities, make sure the following are checked: Bluetooth, Internet (Client & Server) and Remote System.

Building the Session Connection

This app will consist of two pages, with the first page responsible for creating or joining

a remote session using the Remote System APIs. For simplicity, I’ll build this page using the MainPage.xaml that was created with the solution and is already wired into the app to be the first page loaded. The UI has two modes: creating or hosting a session and joining an existing session. Creating a session requires a session name that will be public to users looking to join. Joining an existing session needs to show a list of available nearby sessions. Both modes need a name to be displayed to the user. **Figure 3** shows what the resulting UI should look like for MainPage, and the XAML to build this page can be found in **Figure 4**.

After creating the XAML for the page, create a new ViewModels folder in the app, then add a new public class in that folder called MainViewModel.cs. This view model will be wired into the view to handle the functionality.

The Remote System APIs will discover other machines on the same network, as well as those nearby by using Bluetooth.

The first portion of the view model will handle managing state for the radio buttons that determine if the user is creating a new session or joining an existing one. The state is held in a bool called IsNewSession. Two methods are used to switch the status of this bool, CreateSession and JoinSession:

```
public bool IsNewSession { get; set; } = true;
public void CreateSession()
{
    IsNewSession = true;
}

public void JoinSession()
{
    IsNewSession = false;
}
```

The Checked event for each radio button is bound to one of these methods.

The remaining UI elements are tracked with simple properties. The session name and user name are bound to the SessionName and JoinName properties. The SelectedSession property binds to the SelectedItem property in the ListView and its ItemsSource is bound to the Sessions property:

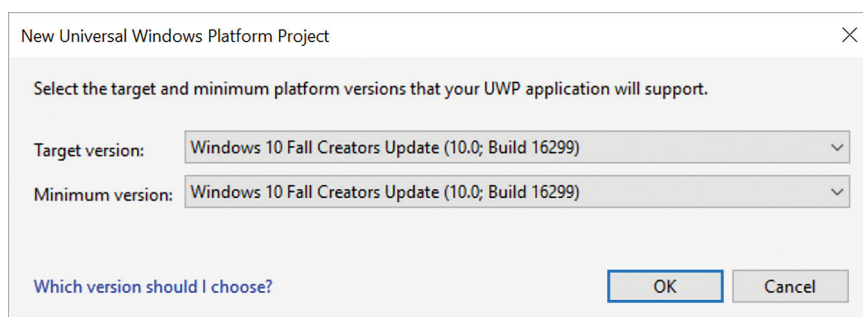


Figure 2 Setting Target Versions for the App

```

public string JoinName { get; set; }
public string SessionName { get; set; }
public object SelectedSession { get; set; }
public ObservableCollection<
    RemoteSystemSessionInfo> Sessions { get; } =
    new -ObservableCollection<
        RemoteSystemSessionInfo>();

```

The view model has two events that will be used to let the view know if a connection to a session was successful or not:

```

public event EventHandler SessionConnected =
    delegate { };
public event EventHandler<SessionCreationResult>
    ErrorConnecting = delegate { };

```

Finally, the Start button is bound to a Start method. This method can be left empty for the moment.

Once the view model has been completed, the codebehind for MainPage needs to create a public property that's an instance of the MainViewModel. This is what allows x:Bind to build the compile-time bindings.

In addition, it needs to subscribe to the two events created in the view model. If a connection is successfully made, I'll navigate to a new page, MessagePage. If the connection fails, a MessageDialog will be displayed, informing the user of the failed connection. Figure 5 contains the code for the MainPage.xaml.cs.

Defining the Data Models

Before digging in the heart of the app, you need to define a couple of data models that will be used in the app. Create a Models folder in the app and then create two classes in it: User and UserMessage. As the name suggests, the User model will track information about the users connected to the app:



Figure 3 The MainPage UI

```

public class User
{
    public string Id { get; set; }
    public string DisplayName { get; set; }
}

```

The UserMessage class will contain the message content, the user creating the content and when the message was created:

```

public class UserMessage
{
    public User User { get; set; }
    public string Message { get; set; }
    public DateTime DateTimeStamp { get; set; }
}

```

Creating a Session

With the code for the main page fairly complete, I can start building out RemoteSessionManager, which will be used to wrap the Remote Systems API. Add a new public class called RemoteSessionManager to the root directory of the app. The app will use a

single shared instance of the RemoteSessionManager, so you can add a static property to the App class in App.xaml.cs:

```

public static RemoteSessionManager SessionManager { get; } = new
    RemoteSessionManager();

```

Before the app can access any of the Remote Systems APIs, it must first obtain permission from the user. This permission is obtained by calling the static method RemoteSystem.RequestAccessAsync:

```

RemoteSystemAccessStatus accessStatus = await RemoteSystem.RequestAccessAsync();
if (accessStatus != RemoteSystemAccessStatus.Allowed)
{
    // Access is denied, shortcut workflow
}

```

The method will return a RemoteSystemAccessStatus enum that can be used to determine if access was granted. This method must

Figure 4 The MainPage XAML

```

<Page
    x:Class="TeamMessenger.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:remotesystems="using:Windows.System.RemoteSystems"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel Width="400"
            HorizontalAlignment="Center"
            BorderBrush="Gray"
            BorderThickness="1"
            MaxHeight="600"
            VerticalAlignment="Center"
            Padding="10">

            <RadioButton x:Name="rbCreate"
                GroupName="options"
                IsChecked="True"
                Checked="{x:Bind ViewModel.CreateSession}"
                Content="Create a New Session"/>

            <StackPanel Orientation="Horizontal" Margin="30,10,20,30">
                <TextBlock VerticalAlignment="Center">Session Name :</TextBlock>
                <TextBox Text="{x:Bind ViewModel.SessionName, Mode=TwoWay}"
                    Width="200"
                    Margin="20,0,0,0"/>
            </StackPanel>

            <RadioButton x:Name="rbJoin"
                GroupName="options"
                Checked="{x:Bind ViewModel.JoinSession}"
                Content="Join Session"/>

            <ListView ItemsSource="{x:Bind ViewModel.Sessions}"
                SelectedItem="{x:Bind ViewModel.SelectedSession, Mode=TwoWay}"
                IsItemClickEnabled="True"
                Height="200"
                BorderBrush="LightGray"
                BorderThickness="1"
                Margin="30,10,20,30">
                <ListView.ItemTemplate>
                    <DataTemplate x:DataType="remotesystems:RemoteSystemSessionInfo">
                        <TextBlock Text="{x:Bind DisplayName}"/>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>

            <StackPanel Orientation="Horizontal">
                <TextBlock VerticalAlignment="Center">Name :</TextBlock>
                <TextBox Text="{x:Bind ViewModel.JoinName, Mode=TwoWay}"
                    Width="200"
                    Margin="20,0,0,0"/>
            </StackPanel>

            <Button Content="Start"
                Margin="0,30,0,0"
                Click="{x:Bind ViewModel.Start}"/>
        </StackPanel>
    </Grid>
</Page>

```

be called from the UI thread so it can successfully prompt the user. Once the user has granted or denied permission to the app, any subsequent calls will automatically return the user's preference. For this app, this permission will be added to the session discovery because it's called first in the workflow.

Note that all of the Remote System APIs can be found in the `Windows.System.RemoteSystem` namespace.

Before the app can access any of the Remote Systems APIs, it must first obtain permission from the user.

The first method to add to the `RemoteSessionManager` class is the `CreateSession` method. Because there are several results that can be returned from this method, I'll wrap those up in a new enum—`SessionCreationResult`. `SessionCreationResult` has four possible values: success and three different failures. A session can fail to create because the user didn't grant access to the app; the app currently has too many sessions running; or a system error failed to create the session:

```
public enum SessionCreationResult
{
    Success,
    PermissionError,
    TooManySessions,
    Failure
}
```

Remote sessions are managed by a `RemoteSystemSessionController`. When creating a new `RemoteSystemSessionController` instance, you must pass in a name that will be displayed to devices attempting to join the session.

Once the controller is requested, a session can be started by calling the `CreateSession` method. This method returns a `RemoteSystemSessionCreationResult` containing a status and the new instance of the session if it was successful. The `RemoteSessionManager` will store the new controller and session in private variables.

Figure 5 MainPage.xaml Codebehind

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        ViewModel.SessionConnected += OnSessionConnected;
        ViewModel.ErrorConnecting += OnErrorConnecting;
    }

    private async void OnErrorConnecting(object sender, SessionCreationResult e)
    {
        var dialog = new MessageDialog("Error connecting to a session");
        await dialog.ShowAsync();
    }

    private void OnSessionConnected(object sender, EventArgs e)
    {
        Frame.Navigate(typeof(MessagePage));
    }

    public MainViewModel ViewModel { get; } = new MainViewModel();
}
```

dtSearch®

Instantly Search Terabytes

dtSearch's **document filters** support:

- popular file types
- emails with multilevel attachments
- a wide variety of databases
- web data

Over 25 search options including:

- efficient multithreaded search
- **easy** **multicolor** **hit-highlighting**
- forensics options like credit card search

Developers:

- APIs for .NET, C++ and Java; ask about new cross-platform .NET Standard SDK with Xamarin and .NET Core
- SDKs for Windows, UWP, Linux, Mac, iOS in beta, Android in beta
- FAQs on faceted search, granular data classification, Azure and more

Visit dtSearch.com for

- hundreds of reviews and case studies
- fully-functional enterprise and developer evaluations

The Smart Choice for Text Retrieval®
since 1991

dtSearch.com 1-800-IT-FINDS

A new public property, `IsHost`, should be added to the manager, as well as to determine workflow. During the `CreateSession` method, this value is set to true, identifying this app as the host. Another public property, `CurrentUser`, provides an instance of the user on the machine and will be used for messaging. The session manager also maintains an `ObservableCollection` of users in the current session. This collection is initialized with the newly created user. For the host of the session, this instance gets created in the `CreateSession` method. The resulting additions to the `RemoteSessionManager` are shown in **Figure 6**.

There are three more items that need to be added to `RemoteSessionManager` to complete the `CreateSession` method. The first is an event handler for when a user attempts to join a session and the `JoinRequested` event is raised on the session. The `OnJoinRequested` method will automatically accept any user trying to join. This could be extended to prompt the host for approval before the user is joined to the session. The request information is provided as a `RemoteSystemSessionJoinRequest` included in the `RemoteSystem-`

Figure 6 The CreateSession Method

```
private RemoteSystemSessionController _controller;
private RemoteSystemSession _currentSession;
public bool IsHost { get; private set; }
public User CurrentUser { get; private set; }
public ObservableCollection<User> Users { get; } =
    new ObservableCollection<User>();

public async Task<SessionCreationResult> CreateSession(
    string sessionName, string displayName)
{
    SessionCreationResult status = SessionCreationResult.Success;

    RemoteSystemAccessStatus accessStatus = await RemoteSystem.RequestAccessAsync();
    if (accessStatus != RemoteSystemAccessStatus.Allowed)
    {
        return SessionCreationResult.PermissionError;
    }

    if (_controller == null)
    {
        _controller = new RemoteSystemSessionController(sessionName);
        _controller.JoinRequested += OnJoinRequested;
    }

    RemoteSystemSessionCreationResult createResult =
        await _controller.CreateSessionAsync();

    if (createResult.Status == RemoteSystemSessionCreationStatus.Success)
    {
        _currentSession = createResult.Session;

        InitParticipantWatcher();

        CurrentUser = new User() { Id = _currentSession.ControllerDisplayName,
            DisplayName = displayName };
        Users.Add(CurrentUser);

        IsHost = true;
    }
    else if (createResult.Status == RemoteSystemSessionCreationStatus.SessionLimitsExceeded)
    {
        status = SessionCreationResult.TooManySessions;
    }
    else
    {
        status = SessionCreationResult.Failure;
    }

    return status;
}
```

`SessionJoinRequestedEventArgs` parameter of the event handler. Invoking the `Accept` method will add the user to the session. The following code includes the new event to add to `RemoteSessionManager`, as well as the completed `OnJoinRequested` method:

```
private void OnJoinRequested(RemoteSystemSessionController sender,
    RemoteSystemSessionJoinRequestedEventArgs args)
{
    var deferral = args.GetDeferral();
    args.JoinRequest.Accept();
    deferral.Complete();
}
```

The session manager can monitor when participants are added or removed from the current session through the `RemoteSystemSessionParticipantWatcher`. This class monitors the participants and raises either an `Added` or `Removed` event when needed. When a user joins a session already in progress, each participant already in the current session will receive an `Added` event. The app will take this series of events and determine which participant is the host by matching the `DisplayName` against the session's `ControllerDisplayName`. This will allow participants to communicate directly with the host. The session manager maintains a participant watcher as a private variable that's initialized in the `InitParticipantWatcher`. This method is called whether creating a session or joining an existing session. **Figure 7** contains the new additions. You'll notice that for this workflow you need to know when a participant is removed only if you're the host, and if a participant is added if you're joining a session. As a host, the `RemoteSessionManager`

Figure 7 InitParticipantWatcher

```
private RemoteSystemSessionParticipantWatcher _participantWatcher;

private void InitParticipantWatcher()
{
    _participantWatcher = _currentSession.CreateParticipantWatcher();
    if (IsHost)
    {
        _participantWatcher.Removed += OnParticipantRemoved;
    }
    else
    {
        _participantWatcher.Added += OnParticipantAdded;
    }

    _participantWatcher.Start();
}

private void OnParticipantAdded(RemoteSystemSessionParticipantWatcher watcher,
    RemoteSystemSessionParticipantAddedEventArgs args)
{
    if (args.Participant.RemoteSystem.DisplayName ==
        _currentSession.ControllerDisplayName)
    {
        Host = args.Participant;
    }
}

private async void
    OnParticipantRemoved(RemoteSystemSessionParticipantWatcher watcher,
        RemoteSystemSessionParticipantRemovedEventArgs args)
{
    var qry = Users.Where(u => u.Id == args.Participant.RemoteSystem.DisplayName);
    if (qry.Count() > 0)
    {
        var dispatcher = CoreApplication.MainView.CoreWindow.Dispatcher;

        await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.High,
            () => { Users.Remove(qry.First()); });

        await BroadcastMessage("users", Users);
    }
}
```


is concerned only when a participant leaves the session. The host will be notified directly by the participant when they join, as you'll see later in the article. Participants only need to determine the current host account.

The session manager maintains a participant watcher as a private variable that's initialized in the `InitParticipantWatcher`.

The last thing the `CreateSession` method needs added to the class is an event to let consumers know if and when the session is disconnected. The new `SessionDisconnected` event can be defined as follows:

```
public event EventHandler<RemoteSystemSessionDisconnectedEventArgs>
    SessionDisconnected =
        delegate { };
```

Joining a Session

Now that the app is able to broadcast a new remote session, the next thing to implement is the ability to join that session from another machine. There are two steps to joining a remote session: discovery and then connecting to the session.

Discovering a Session An app is able to discover nearby remote sessions through the `RemoteSystemSessionWatcher`, which is created from the static `CreateWatcher` method. This class raises events every time a session is added or removed. Add a new method—`DiscoverSessions`—to the `RemoteSessionManager`. This method will create a `RemoteSystemSessionWatcher` as a private variable to the class and handle the `Added` and `Removed` events. These events will be wrapped by two new events added to `RemoteSessionManager`:

Figure 8 Discovering Sessions

```
private RemoteSystemSessionWatcher _watcher;
public event EventHandler<RemoteSystemSessionInfo> SessionAdded = delegate { };
public event EventHandler<RemoteSystemSessionInfo> SessionRemoved = delegate { };

public async Task<bool> DiscoverSessions()
{
    RemoteSystemAccessStatus status = await RemoteSystem.RequestAccessAsync();
    if (status != RemoteSystemAccessStatus.Allowed)
    {
        return false;
    }

    _watcher = RemoteSystemSession.CreateWatcher();
    _watcher.Added += (sender, args) =>
    {
        SessionAdded(sender, args.SessionInfo);
    };

    _watcher.Removed += (sender, args) =>
    {
        SessionRemoved(sender, args.SessionInfo);
    };

    _watcher.Start();

    return true;
}
```



Seeking for professional and easy-to-use Imaging .NET SDK?

VintaSoft Imaging .NET SDK

Load, view, convert, manage, print, capture from camera, save raster images and PDF documents.



- Image Viewer for .NET, WPF and WEB
- 100+ Image Processing and Document Cleanup commands
- PDF Reader, Writer, Visual Editor
- Image Annotations
- JBIG2 and JPEG2000 codecs
- OCR and Document Recognition
- Forms Processing and OMR
- DICOM decoder
- Barcode Reader and Generator
- TWAIN scanning

Free evaluation version
Royalty free licensing

www.vintasoft.com

VintaSoft® is the registered trademark
of VintaSoft Ltd.



Figure 9 Adding Session Discovery

```
public MainViewModel()
{
    _initSessionManager = InitSessionManager();
}

private Task _initSessionManager;

private async Task InitSessionManager()
{
    App.SessionManager.SessionAdded += OnSessionAdded;
    App.SessionManager.SessionRemoved += OnSessionRemoved;
    await App.SessionManager.DiscoverSessions();
}

private async void OnSessionAdded(object sender, RemoteSystemSessionInfo e)
{
    var dispatcher = CoreApplication.MainView.CoreWindow.Dispatcher;

    await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.High,
        () => { Sessions.Add(e); });
}

private async void OnSessionRemoved(object sender, RemoteSystemSessionInfo e)
{
    if (Sessions.Contains(e))
    {
        var dispatcher = CoreApplication.MainView.CoreWindow.Dispatcher;

        await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.High,
            () => { Sessions.Remove(e); });
    }
}
```

SessionAdded and SessionRemoved. Because this will be another entry point for users initializing remote sessions, you need to make sure to add a call to RemoteSystem.RequestAccessAsync. **Figure 8** contains the private variable, the two events and the complete DiscoverSessions method.

It's now possible to wire in the MainViewModel to update the Sessions property with locally available sessions. Because the DiscoverSessions method is asynchronous, the constructor of the MainViewModel needs to initialize a Task to invoke it. The initializing method should also register and handle the SessionAdded and SessionRemoved events. Because these events won't be firing on the UI thread when updating the Sessions property, it's important to use a CoreDispatcher. The updates to MainViewModel are in **Figure 9**.

Connecting to the Session Once a user has identified the session they want to join and supplies a name, the RemoteSessionManager must be able to connect them to the selected session. This will be handled by a new JoinSession method on the RemoteSessionManager, which takes the selected session and the entered display name as parameters.

The JoinSession method begins by calling the JoinAsync method on the supplied session. In return, this will fire the JoinRequested event on the host session. If the host approves the request, a success status is returned and the CurrentUser is set using the display name. As with the CreateSession method, the InitParticipantWatcher method is invoked to register an event handler for when participants are added to the session. The JoinSession method is shown in **Figure 10**.

The final step involved in joining a session is to use the functionality created in the RemoteSessionManager to either create or join a session. **Figure 11** shows the Start method in the MainViewModel that's bound to the Start button in MainPage. The workflow of the method is straightforward. Depending on IsNewSession, it

Figure 10 The JoinSession Method

```
public async Task<bool> JoinSession(RemoteSystemSessionInfo session, string name)
{
    bool status = true;

    RemoteSystemSessionJoinResult joinResult = await session.JoinAsync();

    if (joinResult.Status == RemoteSystemSessionJoinStatus.Success)
    {
        _currentSession = joinResult.Session;
        CurrentUser = new User() { DisplayName = name };
    }
    else
    {
        status = false;
    }

    InitParticipantWatcher();

    return status;
}
```

calls either the CreateSession method or the JoinSession method. The results are returned by raising either the SessionConnected or ErrorConnecting events. If the session is successful, the app navigates to the MessagePage, which I'll build in the next section.

Keep the Apps Talking

At this point, the app can successfully create or join a session and has a messaging UI that's ready to be used. The only remaining step is enabling the devices to communicate with each other. This is accomplished by using the Remote System API to send ValueSet instances between the machines. Each ValueSet is a set of key/value pairs of serialized payloads.

Receiving Messages Messages are transmitted within a session through a RemoteSystemSessionMessageChannel. A session can have multiple channels; however, this app will need only a single channel. In the RemoteSessionManager, I'm going to add a Start-ReceivingMessages method. This method will create a new message

Figure 11 Starting a Session

```
public async void Start()
{
    if(IsNewSession)
    {
        var result = await App.SessionManager.CreateSession(SessionName, JoinName);
        if(result == SessionCreationResult.Success)
        {
            SessionConnected(this, null);
        }
        else
        {
            ErrorConnecting(this, result);
        }
    }
    else
    {
        if(SelectedSession != null)
        {
            var result = await App.SessionManager.JoinSession(
                SelectedSession as RemoteSystemSessionInfo, JoinName);
            if(result)
            {
                SessionConnected(this, null);
            }
            else
            {
                ErrorConnecting(this, SessionCreationResult.Failure);
            }
        }
    }
}
```

Figure 12 Receiving Messages

```
private RemoteSystemSessionMessageChannel _messageChannel;
public event EventHandler<MessageReceivedEventArgs> MessageReceived = delegate { };

public void StartReceivingMessages()
{
    _messageChannel = new RemoteSystemSessionMessageChannel(_currentSession,
        "OpenChannel");
    _messageChannel.ValueSetReceived += OnValueSetReceived;
}

private object DeserializeMessage(ValueSet valueSet)
{
    Type serialType;
    object data;

    if(valueSet.ContainsKey("user"))
    {
        serialType = typeof(User);
        data = valueSet["user"];
    }
    else if (valueSet.ContainsKey("users"))
    {
        serialType = typeof(List<User>);
        data = valueSet["users"];
    }
    else
    {
        serialType = typeof(UserMessage);
        data = valueSet["message"];
    }

    object value;

    using (var stream = new MemoryStream((byte[])data))
    {
        value = new DataContractJsonSerializer(serialType).ReadObject(stream);
    }

    return value;
}

private async void OnValueSetReceived(RemoteSystemSessionMessageChannel sender,
    RemoteSystemSessionValueSetReceivedEventArgs args)
{
    var data = DeserializeMessage(args.Message);

    if (data is User)
    {
        var user = data as User;
        user.Id = args.Sender.RemoteSystem.DisplayName;

        if (!Users.Contains(user))
        {
            var dispatcher = CoreApplication.MainView.CoreWindow.Dispatcher;

            await dispatcher.RunAsync(Windows.UI.Core.CoreDispatcherPriority.High,
                () => { Users.Add(user); });
        }

        await BroadcastMessage("users", Users.ToList());
    }
    else if (data is List<User>)
    {
        var users = data as List<User>;
        Users.Clear();
        foreach(var user in users)
        {
            Users.Add(user);
        }
    }
    else
    {
        MessageReceived(this, new MessageReceivedEventArgs()
        {
            Participant = args.Sender,
            Message = data
        });
    }
}
```

channel that's stored in a private variable and then add a handler to the ValueSetReceived event.

Messages are sent as text and because the app is using classes as messages, I need to serialize the data. When the ValueSet is received from the channel, a DataContractJsonSerializer is used to rehydrate the message classes in the DeserializeMessage class. Because I'm unable to tell what type of message is serialized, the app will send each type of message as a different value in the ValueSet. The DeserializeMessage class will determine which key is used and return the correct class.

Once the message class is ready, the manager class will act on the message depending on its type. As you'll see, participants will announce themselves to the host by sending their CurrentUser instance. In response, the host will broadcast the updated user

list to all participants. If the session manager receives the list of participants, it will update the Users collections with the updated data. The final option, a UserMessage, will raise a new MessageReceived event that passes the message and the participant who sent the message. These additions to the RemoteSessionManager can be found in **Figure 12**.

Figure 12 includes a new event handler class, MessageReceivedEventArgs, that must also be created. This class contains two properties: the sender and the message:

```
public class MessageReceivedEventArgs
{
    public RemoteSystemSessionParticipant Participant { get; set; }
    public object Message { get; set; }
}
```

Figure 13 Broadcasting a Message

```
public async Task<bool> BroadcastMessage(string key, object message)
{
    using (var stream = new MemoryStream())
    {
        new DataContractJsonSerializer(message.GetType()).WriteObject(stream, message);
        byte[] data = stream.ToArray();

        ValueSet msg = new ValueSet();
        msg.Add(key, data);
        await _messageChannel.BroadcastValueSetAsync(msg);
    }

    return true;
}
```

Figure 14 Sending a Direct Message

```
public async Task<bool> SendMessage(string key, object message,
    RemoteSystemSessionParticipant participant)
{
    using (var stream = new MemoryStream())
    {
        new DataContractJsonSerializer(message.GetType()).WriteObject(stream, message);
        byte[] data = stream.ToArray();

        ValueSet msg = new ValueSet();
        msg.Add(key, data);
        await _messageChannel.SendValueSetAsync(msg, participant);
    }

    return true;
}
```

TEXTCONTROL

TX Text Control X15

Automate your reports and create beautiful documents in Windows Forms, WPF, ASP.NET and Cloud applications.

Text Control Reporting combines powerful reporting features with an easy-to-use, MS Word compatible word processor. Users create documents and templates using ordinary Microsoft Word skills.

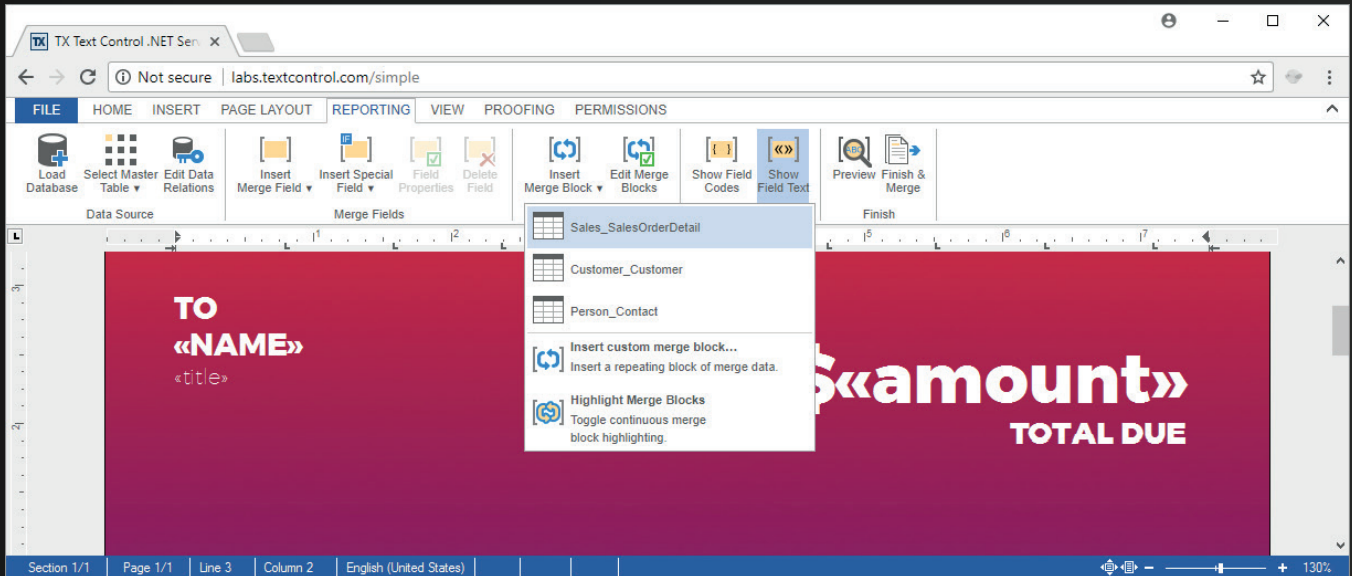
Download a free trial at
www.textcontrol.com



**WE ARE CHANGING
THE WAY YOU LOOK AT
REPORTING**

TX Text Control .NET Server for ASP.NET

Complete reporting and word processing for ASP.NET Web Forms and MVC



✓ Give your users a WYSIWYG, MS Word compatible, HTML5-based, cross-browser editor to create powerful reporting templates and documents anywhere.

✓ Text Control Reporting combines the power of a reporting tool and an easy-to-use WYSIWYG word processor - fully programmable and embeddable in your application.

✓ Replacing MS Office Automation is one of the most typical use cases. Automate, edit and create documents with Text Control UI and non-UI components.



WINDOWS FORMS



WPF



ASP.NET MVC



ASP.NET AJAX



CLOUD WEB API

TEXT CONTROL

Sending Messages The Remote Systems API provides two methods for delivering messages to other users. The first is to broadcast a message to all of the users in the session. This approach will be used for two of our message types, the `UserMessage` and the list

of Users. Let's create a new method, `BroadcastMessage`, in the `RemoteSystemManager`. This method takes a key and the message as parameters. Using a `DataContractJsonSerializer`, I serialize the data and use the `BroadcastValueSetAsync` method to send the message to all of the users, as shown in **Figure 13**.

Figure 15 The MessagePage XAML

```
<Page
  x:Class="TeamMessenger.MessagePage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:TeamMessenger"
  xmlns:models="using:TeamMessenger.Models"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:remoteSystems="using:Windows.System.RemoteSystems"
  mc:Ignorable="d">

  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
      <ColumnDefinition MinWidth="200" Width="Auto"/>
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>

    <Grid VerticalAlignment="Stretch"
      BorderBrush="Gray" BorderThickness="0,0,1,0">
      <ListView ItemsSource="{x:Bind ViewModel.Users}">
        <ListView.ItemTemplate>
          <DataTemplate x:DataType="models:User">
            <TextBlock Height="25"
              FontSize="16"
              Text="{x:Bind DisplayName}"/>
          </DataTemplate>
        </ListView.ItemTemplate>
      </ListView>
    </Grid>

    <Grid Grid.Column="1" Margin="10,10,0,0">
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>

      <ListView x:Name="lvMessages" ItemsSource="{x:Bind ViewModel.Messages}">
        <ListView.ItemTemplate>
          <DataTemplate x:DataType="models:UserMessage">
            <StackPanel Orientation="Vertical"
              Margin="10,20,10,5">
              <TextBlock TextWrapping="WrapWholeWords"
                Height="Auto"
                Text="{x:Bind Message}"/>
              <StackPanel Orientation="Horizontal"
                Margin="20,5,0,0">
                <TextBlock Text="{x:Bind User.DisplayName}"
                  FontSize="12"
                  Foreground="Gray"/>
                <TextBlock Text="{x:Bind DateTimeStamp}"
                  Margin="20,0,0,0"
                  FontSize="12"
                  Foreground="Gray"/>
              </StackPanel>
            </DataTemplate>
          </ListView.ItemTemplate>
        </ListView>

        <Grid Grid.Row="1" Height="60"
          Background="LightGray">
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
          </Grid.ColumnDefinitions>
          <TextBox Text="{x:Bind ViewModel.NewMessage, Mode=TwoWay}"
            Margin="10"/>
          <Button Grid.Column="1" Content="Send"
            Click="{x:Bind ViewModel.SubmitMessage}"
            Margin="10"/>
        </Grid>
      </Grid>
    </Grid>
  </Page>
```

The Remote Systems
API provides two methods
for delivering messages
to other users.

The second approach is to send a message to a single participant. This approach is similar to broadcasting a message, except it uses the `SendValueSetAsync` method to message a participant directly. This final method to the `RemoteSystemManager`, `SendMessage`, can be found in **Figure 14**.

Building the Messaging Page

With the messaging now in place, it's time to put it to use and finish the app. Add a new Blank Page to the app, `MessagePage.xaml`. This page will consist of a list of users, a message window and input fields for adding a message. The full XAML can be found in **Figure 15**.

Like `MainPage`, `MessagePage` will need a view model. Add a new class, `MessageViewModel`, to the `ViewModels` folder. This view model will need to support `INotifyPropertyChanged` to allow the two-way binding to function properly. This view model will contain three properties: `Users`, `Messages` and `NewMessage`. The `Users` will simply expose the `RemoteSessionManager`'s `User` collection to the view. `Messages` will be an `ObservableCollection` of `UserMessage` objects received and a `NewMessage` string containing the text to

Figure 16 MessageViewModel Constructor

```
public event PropertyChangedEventHandler PropertyChanged = delegate { };
public event EventHandler MessageAdded = delegate { };

public ObservableCollection<UserMessage> Messages { get; private set; }

public ObservableCollection<User> Users { get; private set; }

private string _newMessage;
public string NewMessage {
  get { return _newMessage; }
  set
  {
    _newMessage = value;
    PropertyChanged(this, new
      PropertyChangedEventArgs(nameof(NewMessage)));
  }
}

public MessageViewModel()
{
  Users = App.SessionManager.Users;

  Messages = new ObservableCollection<UserMessage>();
  App.SessionManager.StartReceivingMessages();
  App.SessionManager.MessageReceived += OnMessageReceived;
  RegisterUser();
}
```

Figure 17 Submitting a Message

```
public async void SubmitMessage()
{
    var msg = new UserMessage()
    {
        User = App.SessionManager.CurrentUser,
        DateTimeStamp = DateTime.Now,
        Message = NewMessage
    };

    await App.SessionManager.BroadcastMessage("message", msg);
    Messages.Add(msg);

    NewMessage = "";
    MessageAdded(this, null);
}
```

Figure 18 MessagePage Codebehind

```
public sealed partial class MessagePage : Page
{
    public MessagePage()
    {
        this.InitializeComponent();
        ViewModel.MessageAdded += OnMessageAdded;
    }

    private void OnMessageAdded(object sender, EventArgs e)
    {
        lvMessages.ScrollIntoView(ViewModel.Messages.Last());
    }

    public MessageViewModel ViewModel { get; } = new MessageViewModel();
}
```

send as a new message. There is also a single event, MessageAdded, that will be used by the codebehind in MessagePage. In the constructor of the view model, I need to map the Users property, invoke the StartReceivingMessages method in RemoteSessionManager, and register for the MessageReceived event, as shown in Figure 16. The constructor also includes the implementation of INotifyPropertyChanged.

In the constructor there's a call to RegisterUser. This method will send the CurrentUser that was created when joining a session to the host. This announces to the host that a new user has joined and what the display name is. In response, the host will send out the current list of users to be displayed in the app:

```
private async void RegisterUser()
{
    if (!App.SessionManager.IsHost)
        await App.SessionManager.SendMessage("user", App.SessionManager.CurrentUser, App.SessionManager.Host);
}
```

The final piece of the view model is to broadcast a new message from the user. The SubmitMessage method constructs a new UserMessage and calls the BroadcastMessage method on the RemoteSessionManager. It then clears out the NewMessage value and raises the MessageAdded event, as shown in Figure 17.

In the codebehind for MessagePage, shown in Figure 18, I need to do two things: create an instance of the MessageViewModel for the XAML to reference and handle the MessageAdded event. In the event handler I instruct the ListView to scroll to the bottom of the list where the latest message is visible.

Like MainPage, MessagePage will need a view model.

The Team Messaging app should now be ready to run. On one machine run the app and create a new session. Then launch the app on a second machine, which should show the newly created message. Once you join the session you'll be brought to the new message page where you can begin chatting with others in the session, as shown in Figure 19. You have now created a multiuser app using the Remote System API.

Wrapping Up

Creating successful user experiences within apps often requires looking beyond a single device or platform or even user. Microsoft developed Project Rome to enable developers to provide this level of experience within their apps. In this article I built a UWP app using the Remote Systems API; however, by using the Project Rome SDKs available for other platforms, you could extend this app to work on multiple platforms. When building the next great experience for your users, remember to consider how Project Rome can help you make your app more personal. The source code for this article can be found at bit.ly/2FWtCc5. ■

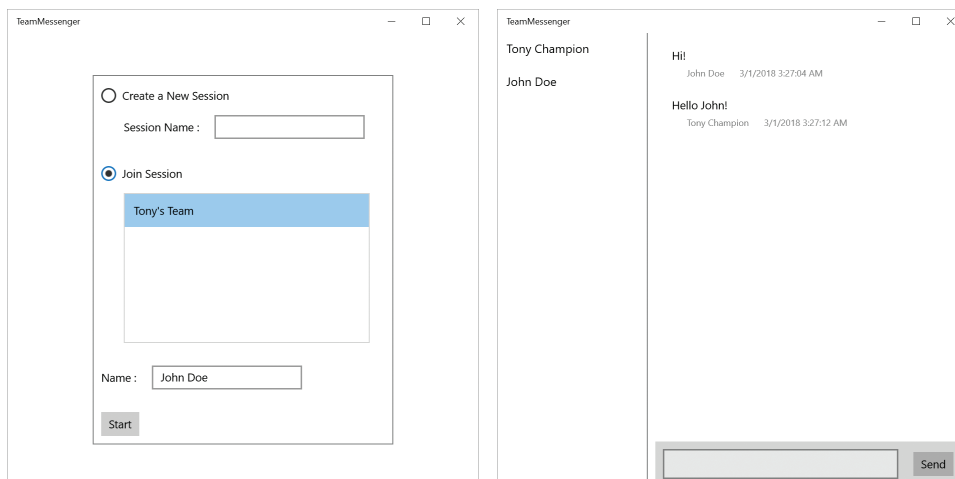


Figure 19 Multiuser Messaging

TONY CHAMPION is a software architect with more than 20 years of experience developing with Microsoft technologies. As the president of Champion DS and its lead software architect, he remains active in the latest trends and technologies, creating custom solutions on Microsoft platforms. His list of clients span multiple industries and includes companies such as: Schlumberger, Microsoft, Boeing, MLB and Chevron/Philips. Champion is an active participant in the community as a six-year Microsoft MVP, international speaker, published author and blogger.

THANKS to the following Microsoft technical expert who reviewed this article: Shawn Henry

AUGUST 6 – 10, 2018 • Microsoft Headquarters, Redmond, WA



Change is ~~Coming~~. Are You Ready? **HERE**

Join us for TechMentor, August 6 – 8, 2018, as we return to Microsoft Headquarters in Redmond, WA. In today's IT world, more things change than stay the same. As we celebrate the 20th year of TechMentor, we are more committed than ever to providing immediately usable IT education, with the tools you need today, while preparing you for tomorrow – **keep up, stay ahead and avoid Winter, ahem, Change.**

Plus you'll be at the source, Microsoft HQ, where you can have lunch with Blue Badges, visit the company store, and experience life on campus for a week!

You owe it to yourself, your company and your career to be at TechMentor Redmond 2018!



SAVE \$400!
REGISTER NOW

Use Promo Code TMMAY

EVENT PARTNER



SUPPORTED BY



PRODUCED BY



AGENDA AT-A-GLANCE

Client and Endpoint Management		PowerShell and DevOps	Infrastructure	Soft Skills for ITPros	Security	Cloud (Public/ Hybrid/Private)
START TIME	END TIME	TechMentor Pre-Conference Workshops: Monday, August 6, 2018 <i>(Separate entry fee required)</i>				
7:30 AM	9:00 AM	Pre-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	M01 Workshop: How to Prevent all Ransomware / Malware in 2018 - Sami Laiho	M02 Workshop: Building Office 365 Federated Identity from Scratch Using AD FS - Nestori Syynimaa		M03 Workshop: Managing Windows Server with Project Honolulu - Dave Kawula	
12:00 PM	2:00 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:00 PM	5:00 PM	M01 Workshop: How to Prevent all Ransomware / Malware in 2018 (Continues) - Sami Laiho	M04 Workshop: Master PowerShell Tricks for Windows Server 2016 and Windows 10 - Will Anderson & Thomas Rayner		M05 Workshop: Dave Kawula's Notes from the Field on Microsoft Storage Spaces Direct - Dave Kawula	
6:30 PM	8:30 PM	Dine-A-Round Dinner - Suite in Hyatt Regency Lobby				
START TIME	END TIME	TechMentor Day 1: Tuesday, August 7, 2018				
7:00 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	T01 Enterprise Client Management in a Modern World - Kent Agerlund	T02 How to Write (PowerShell) Code that Doesn't Suck - Thomas Rayner	T03 The Easy Peasy of Troubleshooting Azure - Mike Nelson	T04 Nine O365 Security Issues Microsoft Probably Hasn't Told You (and You Probably Don't Want to Know) - Nestori Syynimaa	
9:30 AM	10:45 AM	T05 Managing Client Health—Getting Close to the Famous 100% - Kent Agerlund	T06 The Network is Slow! Or is it? Network Troubleshooting for Windows Administrators - Richard Hicks	T07 Getting Started with PowerShell 6.0 for IT Pro's - Sven van Rijen	T08 The Weakest Link of Office 365 Security - Nestori Syynimaa	
11:00 AM	12:00 PM	KEYNOTE: To Be Announced - Stephen L. Rose, Sr. PMM, One Drive For Business, Microsoft				
12:00 PM	1:00 PM	Lunch - McKinley / Visit Exhibitors - Foyer				
1:00 PM	2:15 PM	T09 How to Get Started with Microsoft EMS Right Now - Peter Daalmans	T10 Back to the Future! Access Anywhere with Windows 10 Always on VPN - Richard Hicks	T11 Using Desired State Configuration in Azure - Will Anderson	T12 To Be Announced	
2:15 PM	2:45 PM	Sponsored Break - Visit Exhibitors - Foyer				
2:45 PM	4:00 PM	T13 Conceptualizing Azure Resource Manager Templates - Will Anderson	T14 How to Use PowerShell to Become a Windows Management SuperHero - Petri Paavola	T15 Making the Most Out of the Azure Dev/Test Labs - Mike Nelson	T16 To Be Announced	
4:00 PM	5:30 PM	Exhibitor Reception - Attend Exhibitor Demo - Foyer				
START TIME	END TIME	TechMentor Day 2: Wednesday, August 8, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	W01 Automated Troubleshooting Techniques in Enterprise Domains (Part 1) - Petri Paavola	W02 Troubleshooting Sysinternals Tools 2018 Edition - Sami Laiho	W03 In-Depth Introduction to Docker - Neil Peterson	W04 How Microsoft Cloud Can Support Your GDPR Journey - Milad Aslaner	
9:30 AM	10:45 AM	W05 Automated Troubleshooting Techniques in Enterprise Domains (Part 2) - Petri Paavola	W06 What's New in Windows Server 1803 - Dave Kawula	W07 Simplify and Streamline Office 365 Deployments the Easy Way - John O'Neill, Sr.	W08 How to Administer Microsoft Teams Like a Boss - Ståle Hansen	
11:00 AM	12:00 PM	TECHMENTOR PANEL: The Future of Windows - Peter De Tender, Dave Kawula, Sami Laiho, & Petri Paavola				
12:00 PM	1:00 PM	Birds-of-a-Feather Lunch - McKinley / Visit Exhibitors - Foyer				
1:00 PM	1:30 PM	Networking Break - Exhibitor Raffle @ 1:10 pm (Must be present to win) - Foyer in front of Business Center				
1:30 PM	2:45 PM	W09 Putting the Windows Assessment and Deployment Kit to Work - John O'Neill, Sr.	W10 Deploying Application Whitelisting on Windows Pro or Enterprise - Sami Laiho	W11 Azure is 100% High-Available... Or Is It? - Peter De Tender	W12 What the NinjaCat Learned from Fighting Cybercrime - Milad Aslaner	
3:00 PM	4:15 PM	W13 The Evolution of a Geek—Becoming an IT Architect - Mike Nelson	W14 Advanced DNS, DHCP and IPAM Administration on Windows Server 2016 - Orin Thomas	W15 Managing Tesla Vehicles from the Cloud - Marcel Zehner	W16 Nano Server—Containers in the Cloud - David O'Brien	
6:15 PM	9:00 PM	Set Sail! TechMentor's Seattle Sunset Cruise - Busses depart the Hyatt Regency at 6:15pm to travel to Kirkland City Dock				
START TIME	END TIME	TechMentor Day 3: Thursday, August 9, 2018				
7:30 AM	8:00 AM	Registration - Coffee and Light Breakfast				
8:00 AM	9:15 AM	TH01 Manage Your Apple Investments with Microsoft EMS - Peter Daalmans	TH02 Tips and Tricks for Managing and Running Ubuntu/Bash/Windows Subsystem for Linux - Orin Thomas	TH03 The OMS Solutions Bakery - Marcel Zehner	TH04 Getting Started with PowerShell for Office 365 - Vlad Catrinescu	
9:30 AM	10:45 AM	TH05 HoloLens, Augmented Reality, and IT - John O'Neill, Sr.	TH06 A Real-world Social Engineering Attack and Response - Milad Aslaner	TH07 30 Terrible Habits of Server and Cloud Administrators - Orin Thomas	TH08 Advanced PowerShell for Office 365 - Vlad Catrinescu	
11:00 AM	12:15 PM	TH09 10 Tips to Control Access to Corporate Resources with Enterprise Mobility + Security - Peter Daalmans	TH10 What's New and Trending with Microsoft Enterprise Client Management - Kent Agerlund	TH11 OneNote LifeHack: 5 Steps for Succeeding with Personal Productivity - Ståle Hansen	TH12 Managing Virtual Machines on AWS—Like in Real Life! - David O'Brien	
12:15 PM	2:15 PM	Lunch @ The Mixer - Visit the Microsoft Company Store & Visitor Center				
2:15 PM	3:30 PM	TH13 Security Implications of Virtualizing Active Directory Domain Controllers - Sander Berkouwer	TH14 Building a New Career in 5 Hours a Week - Michael Bender	TH15 Azure CLI 2.0 Deep Dive - Neil Peterson	TH16 OpenSSH for Windows Pros - Anthony Nocentino	
3:45 PM	5:00 PM	TH17 Running Hyper-V in Production for 10 years - Notes from the Field - Dave Kawula	TH18 Network Sustainability and Cyber Security Measures - Omar Valerio	TH19 Azure AD Connect Inside and Out - Sander Berkouwer	TH20 I Needed to Install 80 SQL Servers...Fast. Here's How I Did It! - Anthony Nocentino	
START TIME	END TIME	TechMentor Post-Conference Workshops: Friday, August 10, 2018 <i>(Separate entry fee required)</i>				
8:30 AM	9:00 AM	Post-Conference Workshop Registration - Coffee and Light Breakfast				
9:00 AM	12:00 PM	F01 Workshop: Hardening Your Windows Server Environment - Orin Thomas		F02 Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 - Peter De Tender		
12:00 PM	1:00 PM	Lunch - McKinley				
1:00 PM	4:00 PM	F01 Workshop: Hardening Your Windows Server Environment (Continues) - Orin Thomas		F02 Workshop: Learn the Latest and Greatest Updates to the Azure Platform IaaS and PaaS Services v2.0 (Continues) - Peter De Tender		

Speakers and sessions subject to change

CONNECT WITH TECHMENTOR



@TechMentorEvent

Facebook
Search "TechMentor"LinkedIn
Search "TechMentor"

techmentorevents.com/redmond

Detect and Respond to Rooted Android Devices from Xamarin Apps

Joe Sewell

In last November's issue, I illustrated how you can use Runtime Checks, a code injection feature included with Visual Studio 2017, to protect your .NET Framework apps from unauthorized use of a debugger, as well as from tampering (msdn.com/magazine/mt845626). Since then, a new type of Check has become available. The Root Check detects when a Xamarin.Android app is running on a "rooted" device—one that allows ordinary apps to act with administrator permissions (root access).

In this follow-up article, I explain why rooted devices pose a risk that all Android developers must understand; detail how Xamarin.Android developers can use Root Checks to detect and respond to that risk; and demonstrate best practices with an example scenario.

This article relies on a preview version of Dotfuscator Community Edition version 5.35. All information is subject to change.

This article discusses:

- Why PCI and OWASP advise that Android apps detect and respond to rooted devices
- How to use Root Checks to protect Xamarin apps on Android

Technologies discussed:

Xamarin, Android, Runtime Checks, Dotfuscator Community Edition, Visual Studio 2017

Code download available at:

bit.ly/2GQutOv

Why You Need to Protect Against Rooting

The Xamarin platform allows you to efficiently create mobile apps for Android, iOS and Windows devices. Developers familiar with .NET languages like C# can take that knowledge and apply it to the mobile space. Technologies like Xamarin.Forms abstract away many of the differences between platforms, reducing the complexity, cost and risk of developing cross-platform apps. By keeping your Xamarin tools up-to-date, you can continue to support new versions and features of each platform.

However, some platform-specific aspects of mobile development *do* deserve a developer's attention. One such aspect is security. Each platform has unique security risks and a unique security model to address those risks. For example, the permission systems differ among platforms and sometimes even among versions of the same platform.

For Android apps, rooted devices are a particularly important security concern. Such devices have been modified to allow apps to break out of the normal security sandbox that the OS imposes. This can expose the device to many dangers, such as malware and password-stealing keyloggers. Often, users root their devices to solve some problem—like wanting a version of an app that's not normally available for their device—without realizing the severity of these threats. In other cases, a user may not even be aware that the device is rooted and thus vulnerable.

Last September, the Payment Card Industry Security Standards Council (PCI SSC) issued version 2.0 of the Mobile Payment Acceptance Security Guidelines for Developers. To combat the

security risks associated with rooted devices, the guidelines recommend mobile app developers implement root detection and a response mechanism to quarantine the app (bit.ly/2H5ymge). Here's the relevant text from section 4.3 (emphasis added):

[T]he device should be monitored for activities that defeat operating system security controls—e.g., jailbreaking or **rooting**—and, when detected, the device should be **quarantined** by a solution that removes it from the network, removes the payment-acceptance application from the device, or disables the payment application. **Offline** jailbreak and root detection and auto-quarantine are key since some attackers may attempt to put the device in an offline state to further circumvent detection.

In addition to risks associated with a legitimate user operating the app in a rooted environment, such an environment can also indicate a malicious user attempting to reverse engineer the app. Attackers frequently use rooted devices to study and create tampered versions of apps, which they then fill with malware. The Open Web Application Security Project (OWASP) lists code tampering as one of the Top 10 Mobile Risks (bit.ly/2GNbd4o) and specifically calls out root detection and response as a way to combat this risk. Not doing so, according to OWASP, can lead to reputational damage and lost profits.

Root Checks

Detecting rooted devices can be challenging. A device can be rooted using many different techniques, and the set of available techniques changes over time and across Android versions. As a result, root detection code must constantly evolve and adapt. This is compounded by the fact that some malicious rooting techniques attempt to conceal their use, so good root detection code must also address these countermeasures. Maintaining up-to-date root detection code is tricky and may not be where you want to spend your limited resources.

Luckily, you don't have to write your own code to detect rooting. PreEmptive Protection - Dotfuscator Community Edition (CE), which is included with Visual Studio 2017 for Windows, can inject Root Checks into your Xamarin.Android apps. Root Checks detect rooted environments, even when the device is offline. In addition to a standard "exit the app" action, you can configure the Checks to respond to rooting by calling customized app code.

Just like Xamarin itself, Root Checks reduce complexity, cost and risk compared to rolling your own implementation. Keep Dotfuscator up-to-date and let it handle the root detection—get back to work on the interesting parts of your app quicker.

Sample Scenario

To demonstrate Root Checks, I've provided a sample app called Protected-`TodoAzureAuth`. It's based on an existing Xamarin.Forms sample, `TodoAzureAuth` (bit.ly/2InvU48), originally written by David Britch.

The remainder of this article explains the app, the protection strategy I applied to it, and how I applied that strategy with Root Checks. You can use this case study, as well as additional scenarios included in the sample's GitHub repository (bit.ly/2GQut0v), to learn approaches to Root Checks you can then apply to your own Xamarin.Android apps.

Original Sample: `TodoAzureAuth` connects to a Microsoft Azure Mobile App instance, enabling users to view and modify a shared to-do list. To demonstrate how to perform authentication in a Xamarin app, the sample requires the user to log in with a Google account before accessing the to-do list.

The app begins on the Login Page, which has no fields, just a Login button. When the user selects this button, the app delegates the login process to Google's OAuth system, which may require the user to enter credentials, including a password. As a result, the app itself doesn't handle the credentials. Once the user has logged in, the app displays the Todo List Page, allowing the user to access the shared to-do list. The user can log out and return to the Login Page by selecting the Logout button.

Protection Strategy: For this article, I treated the `TodoAzureAuth` Android project, `TodoAzure.Droid`, as if it were handling sensitive data, like a PCI-compliant app would. I implemented an appropriate protection strategy by using Dotfuscator CE to inject a Root Check into the app, producing a protected version of the app, Protected-`TodoAzureAuth`.

In the protected app, when the user selects the Login button, the Root Check activates. If the app is running on a rooted device, it exits abruptly, and all further attempts to run the app will also exit after a short error message, even if the device is no longer rooted. **Figure 1** shows an overview of the app protected by this strategy.

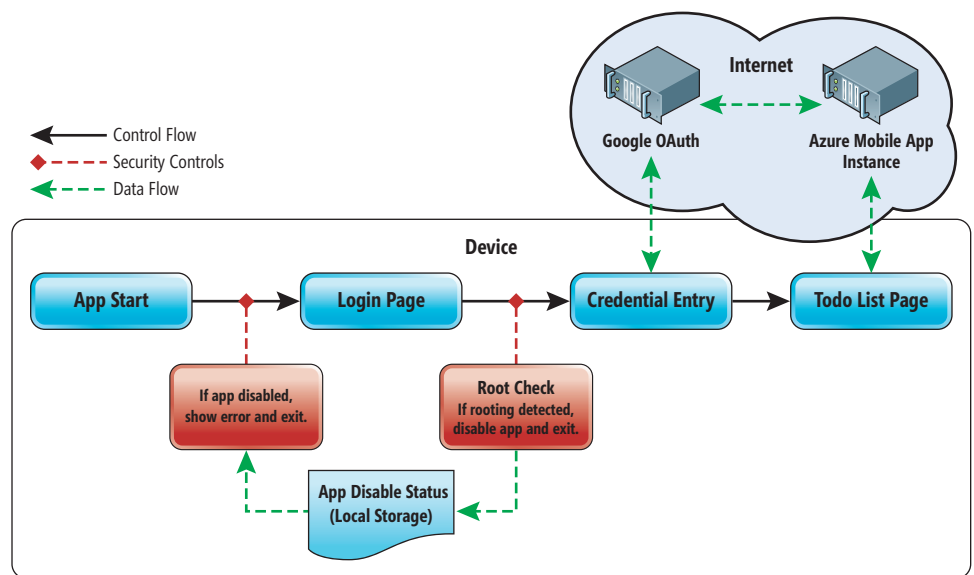


Figure 1 Overview of the Protected-`TodoAzureAuth` Sample App

This strategy aligns with the recommendations made by the PCI guidelines quoted earlier:

- The app monitors the device for rooting.
- The app quarantines the device by disabling itself if rooting is detected.
- This security control operates even when the device is offline.

When the app disables itself, the error message alerts the user that the device is unsafe. While not used in the sample, an app using this scenario could also “phone home” to an analytics platform such as Visual Studio App Center (bit.ly/2pYMuk5).

In addition to following the PCI guidelines, this strategy also lines up with the OWASP recommendation to shut down the app in a rooted environment to prevent reverse engineering. I configured the Root Check to activate at other parts in the code, not just the Login process, so that if an attacker produces a tampered version of the app with the Login process's root detection removed, other parts of the app can still react to rooting. Dotfuscator also obfuscated the code, adding another layer of protection to the app and the Root Check.

Not all apps have the same security requirements, and thus not all apps should react to rooting in the same way. I chose a strict approach for the sample, but a more lenient strategy could allow the app to run on rooted devices in certain circumstances. For an example, see “An Alternate Protection Strategy” included with the Web version of this article at msdn.com/magazine/mt814987.

Protected Sample: You can view the Protected-`ToDoAzure.Auth` sample using the GitHub link supplied earlier. On the default master branch, I've already configured Dotfuscator CE to protect `ToDoAzure.Droid` with a Root Check so that the app meets the strategy explained earlier. You can follow the Git history, starting with the `before-checks` branch, to see how I applied the steps in this article to the sample.

Not all apps have the same
security requirements, and thus
not all apps should react to
rooting in the same way.

Please see the sample's README for details on how to set up, build and run the sample. The README also includes details on other branches present in the repository that demonstrate different protection strategies than the one used for this article, such as the strategy detailed by the aforementioned Web version of this article.

Integrating Dotfuscator into Xamarin Builds

Because Dotfuscator operates on .NET assemblies (.dll and .exe files), not mobile platform formats like Android Packages (.apk files), I had to integrate Dotfuscator into the Xamarin build process. Setting up the integration requires installing and registering Dotfuscator CE, downloading a specialized MSBuild targets file and modifying the project file to include those targets. I've written on how to perform

these integration steps for the Xamarin Blog, so please see those instructions at bit.ly/2w9em6c.

Important Note: Root Checks require Dotfuscator CE version 5.35 or later for Visual Studio 2017. You can always get the latest version at bit.ly/2fuUeow.

I followed the Xamarin Blog instructions to protect the `ToDoAzure.Droid` project file's Release | AnyCPU build configuration. This article only concerns this Android project because Root Checks are an Android-specific feature, but you can also follow the Xamarin Blog instructions to protect iOS and Universal Windows Platform (UWP) projects with code obfuscation.

Configuring the Dotfuscator Protection

Once I integrated Dotfuscator into the `ToDoAzure.Droid` project's build process, I configured the protection through the Dotfuscator CE UI. The protection settings for a project are saved in a specialized Dotfuscator config file, which the build integration adds to your project the first time it builds.

Creating the Dotfuscator Config File: Using Visual Studio 2017, I built the `ToDoAzure.Droid` project in the Release build configuration for the AnyCPU platform, which is the configuration I had set up to use Dotfuscator. This produced a new Dotfuscator config file, `DotfuscatorConfig.xml`, in the project's directory. I added this new file to source control so I could later customize and reapply the protection based on that customization.

The build also created a `DotfuscatorReports` directory in my project directory, which is where Dotfuscator writes various report files when it runs as part of the integration. Because this directory's contents update every build, I had my source control ignore this directory.

Opening Dotfuscator: To customize the Dotfuscator config file, I opened the Dotfuscator CE UI from Visual Studio 2017 by choosing Tools | PreEmptive Protection - Dotfuscator. In the Dotfuscator UI that appeared, I chose File | Open Project, navigated to the `ToDoAzure.Droid` project's directory and selected `DotfuscatorConfig.xml`, the Dotfuscator config file. The Dotfuscator UI updated to display the two assemblies this Dotfuscator config file protects: `ToDoAzure.Droid.dll` itself and the portable class library (PCL) assembly `ToDoAzure.dll`.

Keep in mind that the Build Project option in the Dotfuscator UI doesn't perform Xamarin packaging steps. To ensure the Android Package contains the protected assemblies, build the project from Visual Studio or MSBuild instead.

Enabling Code Injection: Checks are part of the Dotfuscator code injection features. To enable code injection, I right-clicked the Injection node in the Dotfuscator navigation bar and checked the Enable option. The Injection node's text color changed from gray to black, indicating that injection was enabled.

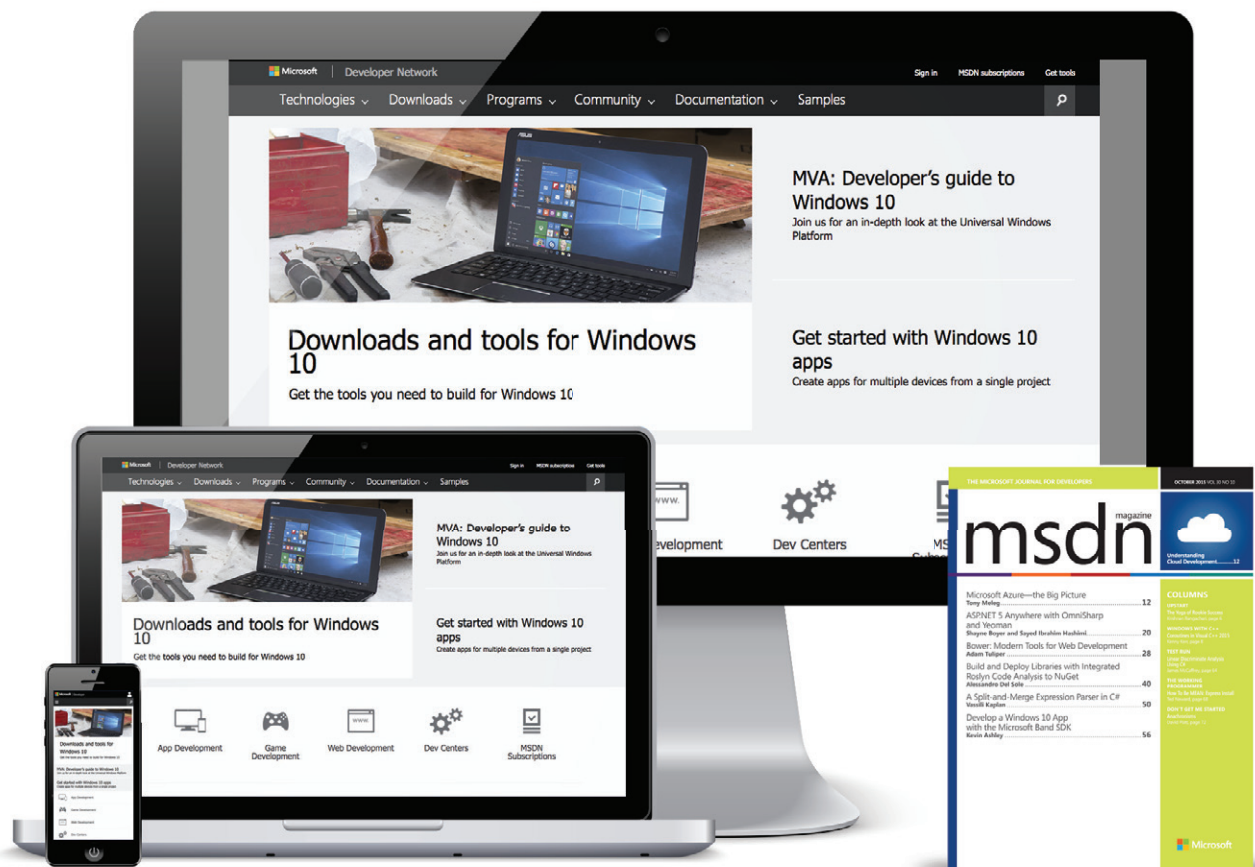
Viewing Configured Checks: The Dotfuscator Checks page displays a list of all configured Checks for a loaded config file, in this case `DotfuscatorConfig.xml` in the `ToDoAzure.Droid` project. To view this page, I selected the Injection node and switched to the Checks tab.

When I first visited this list, it was empty. Once I configured a new Root Check, as I explain in the next section, the list updated

msdn

magazine

Where you need us most.



Visual Studio **LIVE!**
EXPERT SOLUTIONS FOR .NET DEVELOPERS

LIVE!
360
TECH EVENTS WITH PERSPECTIVE

MSDN.microsoft.com

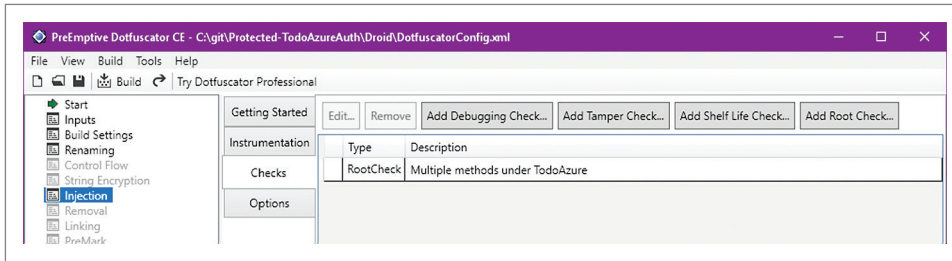


Figure 2 The Dotfuscator Checks Page, Showing a Root Check

to include a row for that Check, as seen in **Figure 2**. I could view the configuration for the Check by double-clicking that row.

Note that you can configure more than one Root Check for a single Dotfuscator config file, though I didn't do so for this article. For an example of an app protected by multiple Checks, see the Adventure-WorksSalesClient .NET Framework app I wrote about last November.

Adding a Root Check

From the Checks page, I added a Root Check by clicking the Add Root Check button. When I did, Dotfuscator displayed a new window for configuring the new Check. **Figure 3** shows the finished configuration; this section explains the meaning of each setting and why I chose those settings.

Checks can report the results of their detection to app code. This allows you to have customized reporting and response behaviors ...

Locations: Each Check is associated with one or more methods in the app, called Locations. When the app calls such a method, the Root Check activates, detecting at that moment if the device appears to be rooted. After executing all configured reporting and response functionality, assuming those measures didn't exit the app, the Check returns control to the top of the method.

For this scenario's Check, I selected multiple Locations. The Location first used in the app is `TodoAzure.Droid.MainActivity`, `AuthenticateAsync`, which coordinates a login request. Using this location means the Root Check will perform its detection and response whenever the login process begins.

Per the protection strategy, an app running on a rooted device exits when it first reaches the `AuthenticateAsync` method. So

Some of these additional Locations are defined in `TodoAzure.dll`. This can be surprising, as that assembly contains logic common to all Xamarin platforms, not just Android. How can Dotfuscator inject a Root Check—which detects rooted Android devices—into a platform-agnostic assembly? Recall that this Dotfuscator config file is specific to the `TodoAzure.Droid` project, which references the `TodoAzure` project. When Dotfuscator modifies `TodoAzure.dll`, it will modify only the assembly that Visual Studio or MSBuild copies for use in the current project, `TodoAzure.Droid`. The original `TodoAzure` project's assembly remains unchanged.

Application Notification: Checks can report the results of their detection to app code. This allows you to have customized reporting and response behaviors, while having the Checks injected by Dotfuscator handle the detection work. The app code that receives the detection result is called an Application Notification Sink.

To meet the protection strategy in this scenario, I needed to have the app disable itself, so future runs of the app exit with an error message. I chose to add this disabling logic in a method, `TodoAzure.App.DisableIfCompromised`, and use it as the Check's Sink by setting the following Check Properties:

- *ApplicationNotificationSinkElement*: The kind of code element; in this case, a method.
- *ApplicationNotificationSinkName*: The simple name of the code element; in this case, `DisableIfCompromised`.
- *ApplicationNotificationSinkOwner*: The type that contains the code element; in this case, `TodoAzure.App`.

Any of the Check's Locations can call this Sink method as it's public and static. To be compatible with a Check, the method is synchronous (non-async), takes a single bool argument and has a void return type.

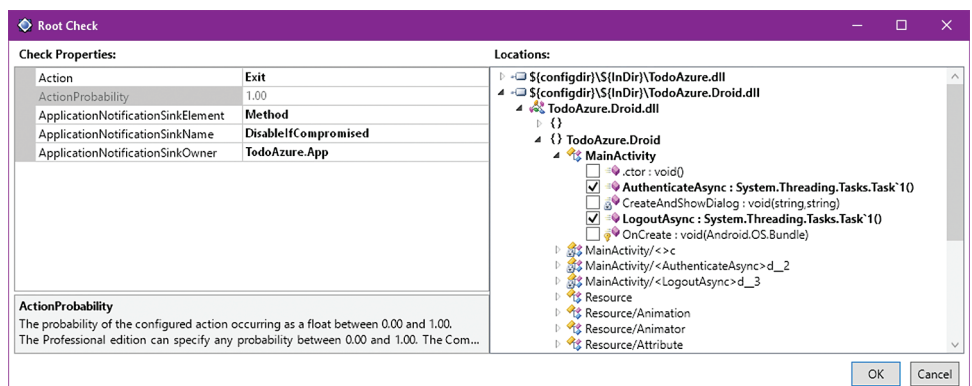


Figure 3 Configuration of the Root Check—Additional Locations Hidden by the Collapsed `TodoAzure.dll` Node



Rider

New .NET IDE

**Cross-platform.
Powerful.
Fast.**

From the makers of ReSharper,
IntelliJ IDEA, and WebStorm.

Learn more
and download
jetbrains.com/rider



**JET
BRAINS**

When activated, the Check calls the method, passing in the argument *true* if the device is rooted and *false* otherwise. When this argument is *true*—that is, when the Check detects rooting—the method saves a value to local storage, indicating the app is now disabled. An accompanying property, *IsDisabled*, exposes the saved value:

```
// Definitions in TodoAzure.App
private const string DisabledPropertyKey = "AppStatus";
public static void DisableIfCompromised(bool wasCompromised)
{
    if (!wasCompromised) { return; }
    Current.Properties[DisabledPropertyKey] = new Random().Next();
    SavePropertiesNow();
}

public static bool IsDisabled =>
    Current.Properties.ContainsKey(DisabledPropertyKey);
```

After the app is disabled, future runs need to show an error message and exit. To do this, I overrode *TodoAzure.LoginPage.OnAppearing*, which is called right before the Login Page is shown when the app starts. If the app is disabled, this method hides the Login Page, displays an error dialog and then exits.

```
// Definition in TodoAzure.LoginPage
protected override async void OnAppearing()
{
    if (App.IsDisabled)
    {
        IsVisible = false;
        var message = "The security of this device has been compromised. "
            + "The app will exit.";
        await DisplayAlert("App deactivated", message, "Exit App");
        App.Exit(); // Delegates to platform-specific exit logic
    }
    base.OnAppearing();
}
```

Because I also want to defend against reverse engineering, I took additional measures to ensure the app would be more resilient to such an attack. I used a vague name for the saved value, *AppStatus*, and set the value to a random number, which obscures the meaning of the value. I also configured Dotfuscator to obfuscate the app, renaming identifiers like *DisableIfCompromised*, so an attacker viewing decompiled code will not easily identify this method as being of interest. For details on how I configured renaming obfuscation, see the sample's README.

When the Check detects a rooted device, it notifies the Sink and then immediately exits the app.

Action: While the Sink (the *DisableIfCompromised* method) sets a property to ensure future runs of the app exit, it doesn't itself exit the app when rooting is first

detected. Instead, I configured the Check to do this automatically by setting the *Action* Check Property to *Exit*.

When the Check detects a rooted device, it notifies the Sink and then immediately exits the app. By having the Check, rather than the Sink, perform this initial exit, I spread multiple copies of the exit logic through the app. Just as with multiple Locations, multiple copies of the exit logic allow the app to better defend itself when an attacker has removed some of the Root Checks.

Building and Testing the App

After configuring the Root Check, I exited the Check's window by selecting OK, then I saved my changes to the Dotfuscator config file by choosing File | Save Project. I built *TodoAzure.Droid* in Visual Studio to test the protected app, in order to verify that I correctly configured the Root Check to enforce the intended protection strategy.

I tested the app on a non-rooted device, on a rooted device and on an emulator. On the non-rooted device, the app functioned normally, allowing me to log in to view the to-do list. However, on the rooted device and on the emulator, after selecting the Login button, the app abruptly closed. After re-launching the app, the app displayed the error dialog shown in **Figure 4**; after I closed the dialog, the app exited once more. To view the Login Page again, I had to uninstall and then reinstall the app.

Wrapping Up

I hope this article has helped illuminate a way to effectively detect and respond to rooted Android devices using free tooling included with Visual Studio. While I used a well-known sample app as reference, you can apply the ideas introduced in this article to all kinds of Xamarin.Android apps and to various other protection strategies.

If you're interested in learning more about Checks, I recommend reading my previous *MSDN Magazine* article. In it, I discussed additional kinds of Checks that you can apply to .NET Framework apps and how using Checks can prevent data breaches.

You may also be interested in the advanced Check and obfuscation features of Dotfuscator Professional Edition (bit.ly/2xgEZcs) or the companion tool for Java and traditional Android apps, PreEmptive Protection - DashO (bit.ly/2ffHTnN). You can keep up-to-date with all developments in Checks and PreEmptive Protection by following PreEmptive Solutions on Twitter (twitter.com/preemptive) and by visiting our blog (preemptive.com/blog). ■

JOE SEWELL is a software engineer and technical writer on the Dotfuscator team at PreEmptive Solutions. He has previously written for MSDN Magazine and the official Xamarin Blog.

THANKS to the following Microsoft technical expert for reviewing this article: David Britch

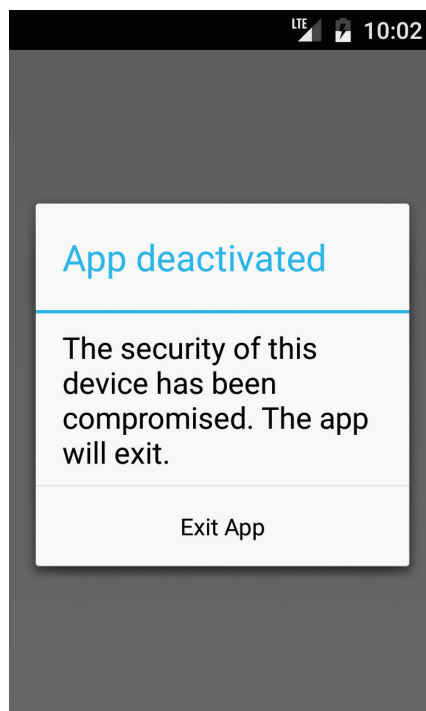


Figure 4 The Protected *TodoAzure.Droid* Running in an Emulator

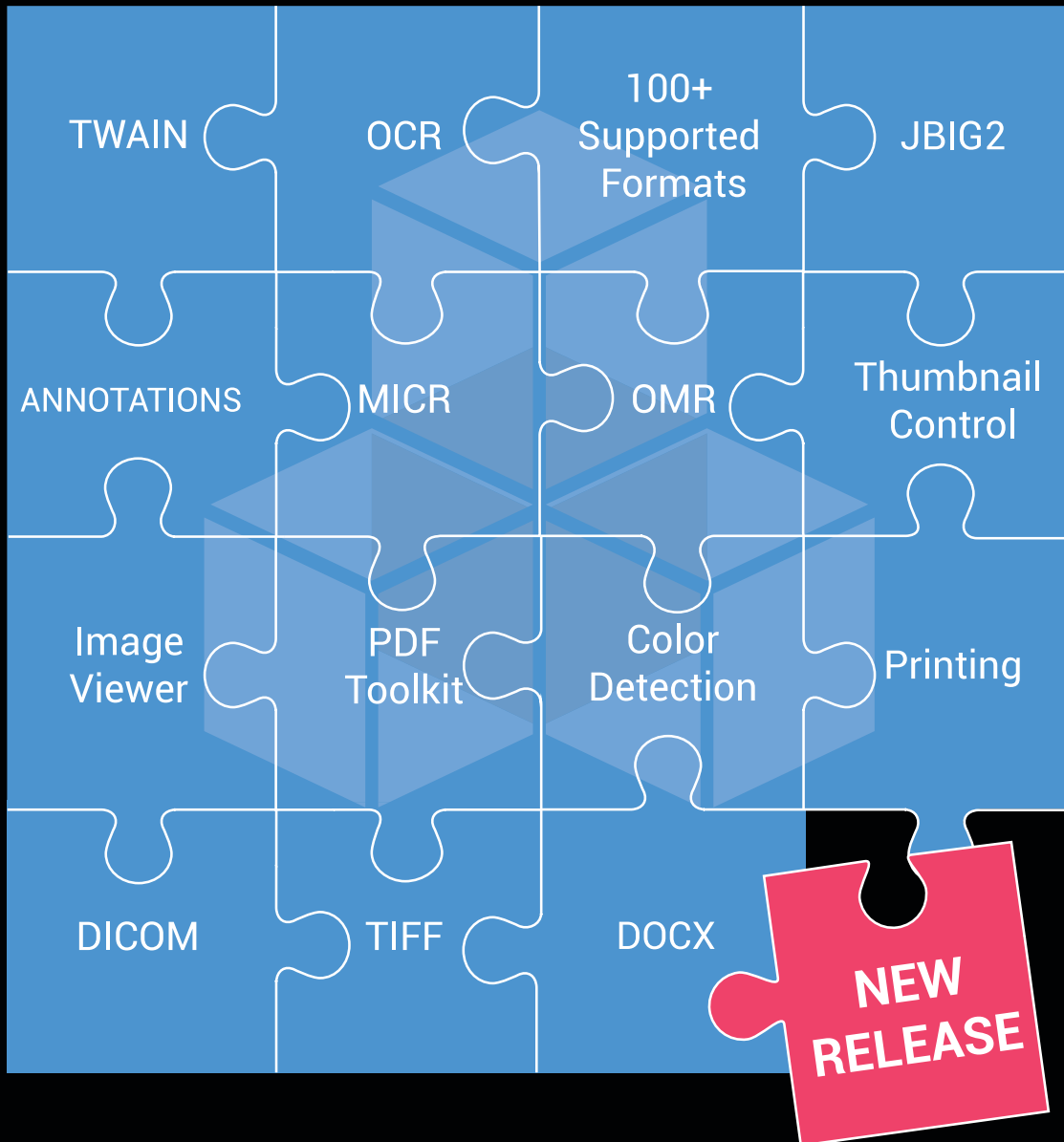
GdPicture.NET



14

100% ROYALTY FREE

Imaging SDK For WinForms, WPF And Web Development



Leverage your apps. with **GdPicture.NET** Imaging Toolkit

DOWNLOAD
YOUR FREE TRIAL

www.gdpicture.com

GdPicture.NET is an



product

Any Language, Any Platform with Azure DevOps Projects

Willy-Peter Schaub and Alex Mullans

Over the past two years, Microsoft has rolled out an ambitious continuous integration (CI) and continuous delivery (CD) pipeline initiative for its DevOps platform. The ALM | DevOps Rangers team has been active in the effort, covering the creation of and learnings from each release, both in numerous blog posts and in *MSDN Magazine* articles such as “Applying DevOps to a Software Development Project” (msdn.com/magazine/mt767695) and “The Road to Continuous Delivery with Visual Studio Team Services” (msdn.com/magazine/mt814804). Years of investment are paying dividends, as Microsoft today boasts a fast-paced and heterogeneous DevOps ecosystem built around a Visual Studio Team Services (VSTS) platform that has evolved by leaps and bounds.

Behind this effort are development teams that demand built-in support for popular application frameworks, automatic and seamless CI/CD pipeline integration, and built-in monitoring support—all in the language and platform of their choice. In addition, developers

want to build any application, on any Azure service, in minutes. Speed and simplicity are paramount.

Microsoft Azure DevOps Projects (bit.ly/2E5zkWG) addresses all these needs. It enables developers to launch an app on any Azure App Service in just a few quick steps, providing everything needed to develop, deploy and monitor an app. Create a DevOps Project, and it provisions all the Azure resources and provides a Git code repository, Application Insights integration and a continuous delivery pipeline setup for deployment to Azure. The DevOps Project dashboard lets you monitor code commits, builds and deployments from a single view in the Azure portal.

This article explores how DevOps Projects enable cloud-powered CI/CD using VSTS to:

- Get up and running with a new application and a full DevOps pipeline in just a few minutes
- Support a wide range of popular frameworks such as .NET, Java, PHP, Node.js and Python
- Integrate with built-in Application Insights for instant analytics and actionable insights
- Start fresh or bring an existing application over from GitHub

Alternatively, we could look at the DevOps Project as a feature to help users “get started” on Azure. Imagine a developer who needs to develop and deploy a Node.js solution to Azure. The user needs to find which Azure service works best for the language, framework and platform scenario, prototype the environment, and finally create an engineering process for the team. Each step has a learning

This article discusses:

- Creation of a continuous integration/continuous delivery (CI/CD) pipeline
- Explores the Azure and VSTS magic (power) behind the scenes
- How to tweak and enhance the CI/CD pipeline

Technologies discussed:

Visual Studio Team Services, Azure DevOps, Package Management

Figure 1 Application, Framework and Deployment Options

Application	Framework	Windows Deployment		Linux Deployment	
		Web App	Virtual Machines	Web App	Web App for Containers
.NET	ASP.NET	X	X		
	ASP.NET Core	X	X	X	X
Java	Spring	X			X
	JSF	X			X
Node.js	Express.js	X			X
	Sails.js				X
	Simple Node.js	X			X
PHP	Laravel	X			X
	Simple PHP	X			X
Python	Django	X		X	
	Bottle	X		X	
	Flask	X		X	X

curve and multiple failure points. The DevOps Project simplifies all this and makes it failure-proof. It's not just about DevOps, but also about making it simpler to get started on Azure.

Discussing DevOps

I'd like to pause for a moment to reiterate our definition of DevOps here at Microsoft. As Donovan Brown, principal DevOps manager in the Azure group describes it: "DevOps is the union of people, process and products to enable continuous delivery of value to our end users."

You can also peruse the DevOps at Microsoft site (aka.ms/devops) for insight into the lessons learned by the Microsoft Developer Division during our seven-year journey to Agile DevOps. That effort has produced a culture here that's focused on the customer, puts production first and continuously delivers value on a three-week cloud cadence.

In this article we'll focus on the easier part of our DevOps definition, the products. As mentioned, developers crave a process that's easy, quick and consistent. They demand support for their choice of programming language and platform, as well as for automation and out-of-the-box CI/CD pipelines.

A CI/CD pipeline is a sequence of distinct practices used to continuously integrate, test and deploy your changes to production. Insights from telemetry, user feedback, and live site incidents flow back to inform decision making, improve future releases, and most important, evaluate the hypothesis that motivated the deployment. It's about continuously delivering value.

Azure DevOps Projects

With Azure DevOps Projects, you can build an Azure application, on an Azure service, in minutes. You also get automatic full CI/CD pipeline integration, built-in monitoring and deployment to the platform of your choice. As shown in the table in **Figure 1**, there are several application, framework and deployment types to choose from when creating a new DevOps Project. It's important to note that this table reflects information at the time of writing and that built-in support will have improved by the time you read this.

Let's explore how to create a new DevOps Project on the Azure portal by creating a pair of applications. I recommend that you get started with Azure DevOps Projects and follow along using your own Azure and VSTS accounts. This approach will let you explore the resulting solutions in detail.

The first application example is based on Java and the Spring framework, and has been deployed as a Web app on

Windows. It's one of many possible examples that shows a mix of technologies running on Windows—a common scenario for developers.

In the Azure portal you create a new DevOps Project. You can then choose from a set of sample apps developed using .NET, Node.js, PHP, Python or Java, or you can bring our own code.

With Azure DevOps Projects, you can build any Azure application, on any Azure service, in minutes.

To get started, select the Java sample application, choose Spring as your application framework and select Web App on Windows as your Azure Service to deploy the app. Next, enter a new project name and confirm or change the Azure and VSTS details. You'll create a new VSTS account and change the App Service location to Canada Centre. Similarly, you can change Azure settings like subscription, pricing plan, Web App name and the location of Web App. Built-in validations ensure that the right values are entered and that downstream failures are avoided.

Once complete, you repeat the exercise, but this time select .NET, .NET Core Framework and the Web App for Containers on Linux to deploy the application. This is another simple example that shows how a mix of technologies can run in a container on Linux. By completing these four simple steps, you've created two DevOps projects, as shown in **Figure 2**.

These projects let you easily and quickly create complete CI/CD pipelines for several popular languages, frameworks and platforms. This is a useful asset for quick demonstrations and experimental projects, and is a great way to get your development team started with a comprehensive, extendible, and consistent process.

The screenshot shows the 'DevOps Projects' interface in the Azure portal. At the top, there are tabs for 'Add', 'Edit columns', 'Refresh', and 'Assign Tags'. Below this, a section titled 'Subscriptions: 1 of 14 selected - Don't see a subscription? Switch directories' contains a search bar and several dropdown menus for filtering by name, resource group, location, and grouping. A table below shows 2 items. The table has columns for 'NAME', 'ACCOUNT', and 'SUBSCRIPTION'. Two items are listed: 'dNCoWeA4CDemo' and 'JaSpWeADemo', both associated with 'MSDNMagazineDemo' accounts and subscriptions. Each item has a checkbox and a three-dot menu icon.

NAME	ACCOUNT	SUBSCRIPTION
dNCoWeA4CDemo	MSDNMagazineDemo	MSDN Magazine Demo
JaSpWeADemo	MSDNMagazineDemo	MSDN Magazine Demo

Figure 2 DevOps Projects

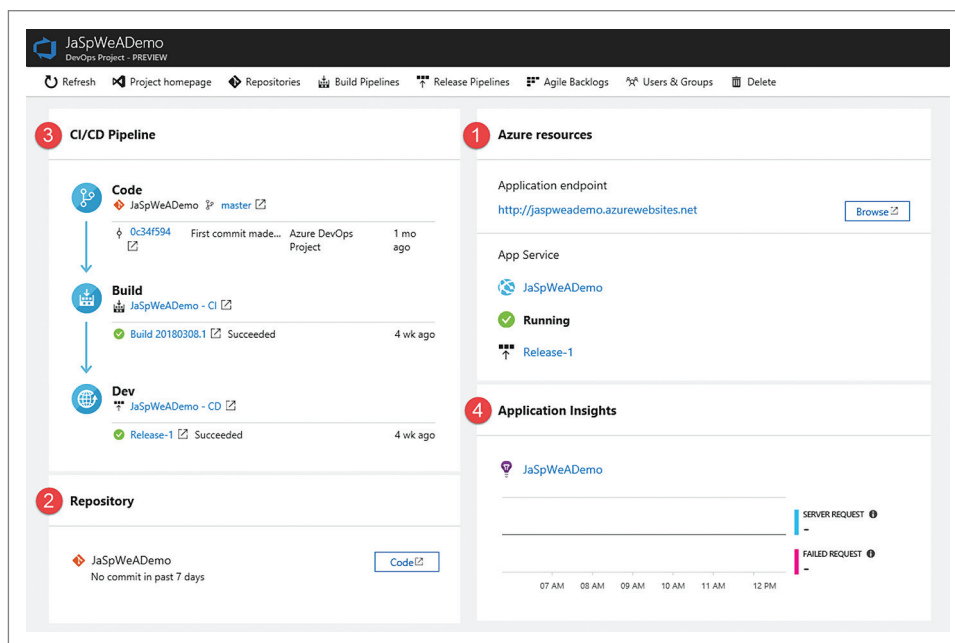


Figure 3 Pipeline View on Azure

Magic Behind the Scenes

Obviously, none of this happened by magic. When you clicked the Done button after selecting the runtime framework, service and configuration, it kicked off a sequence of distinct stages, shown in **Figure 3**. You can navigate to the shown DevOps Project dashboard by selecting the new project in the list of DevOps projects or using the notifications on the Azure portal.

The four stages involved are:

1. **Azure resources:** Created and configured your Azure App Service resources, App Service plan, and application endpoint.
2. **Repository:** Created a distributed Git repository and checked in sample code.
3. **CI/CD Pipeline:** Seamlessly connected with the VSTS collaboration solution for engineering teams, allowing you to plan, test, release and monitor your solutions.
4. **Application Insights:** Created and configured your Application Insights telemetry. This tool enables active monitoring and learning to proactively detect issues and continuously analyze and test hypotheses without code.

Note that the DevOps Project stages not only created and configured a CI/CD Pipeline (see section 3 in **Figure 3**), they also performed a pull of your Code, a continuous integration Build and deployed the release to the Dev environment.

From the DevOps Project dashboard, monitor your CI/CD pipelines, Application Insights telemetry, and navigate to the relevant Azure or VSTS resources.

When you navigate to your new VSTS account, you'll find two new Projects: JaSpWeADemo and dNCoWe4DDemo (**Figure 4**). These are linked to the two Azure DevOps Projects.

In the Build, CI runs a Maven-based build whenever someone checks in code. To reduce the number of running builds, changes are batched when a build is running. You can fine-tune the events that trigger a build, for example specifying branches or using path filters to reduce the set of files that can trigger action.

When you explore the JaSpWeADemo project Pipeline, as shown in **Figure 5**, you're presented with (1) Artifacts and (2) Environments. The Drop artifact is a deployable part of your application produced by the Build. The Continuous deployment trigger instructs Release Management to create a new release when it detects that a new artifact is available. Again, you can fine-tune the events that

trigger the release, for example only processing artifacts from the master branch.

By default, the CI/CD pipeline created by the DevOps Project automatically deploys to the Dev environment when a release is created by a continuous deployment trigger. The Dev environment requires no manual approval, Gates are disabled, and it has only one task that deploys the sample Azure App Service.

The pipeline generated by the DevOps Project is a great starting point for demonstrations and experimentation. For production, the pipeline may require some additional love, which we'll discuss in the next section.

Last, open the second dNCoWe4DDemo sample project and have a look at the CI/CD pipeline. Overall the pipeline looks very similar. However, drill into the build and you'll notice that the CI build triggers a build of the container image, pushing it to the Azure Container Registry. The CD release deploys the Azure App Service on Linux, without any Linux or container expertise required.

As shown in this quick walk-through, DevOps Projects get you started quickly with a complete and functional CI/CD pipeline. The learning curve for Azure, VSTS and DevOps is significantly lower when using DevOps Projects, compared to building the solution and the CI/CD pipeline by hand. You're now empowered to work

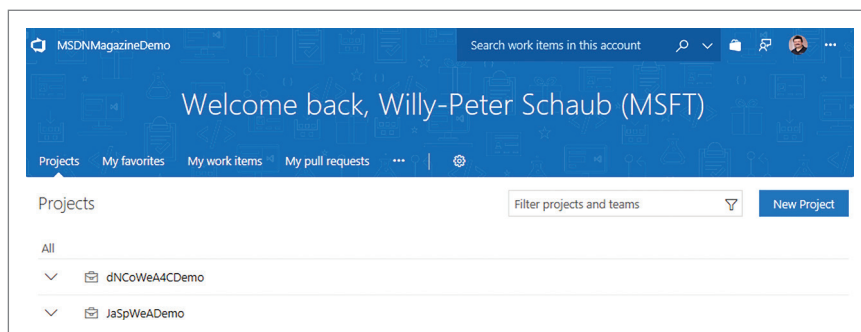


Figure 4 Visual Studio Team Services Account and Team Projects

Manipulating Documents?

APIs to view, convert, annotate, compare, sign, assemble and search documents in your applications.

Try GroupDocs APIs for FREE



GROUPDOCS
Document Manipulation APIs



GroupDocs.Viewer

View over 50 documents and image formats in any application using document viewer APIs.



GroupDocs.Annotation

Add annotations to specific words, phrases and any region of the document.



GroupDocs.Conversion

Fast batch document conversion APIs for any .NET, Java or Cloud app.



GroupDocs.Comparison

Compare two documents and get a difference summary report.



GroupDocs.Signature

Digitally sign Microsoft Word, Excel, PowerPoint and PDF documents.



GroupDocs.Assembly

Document automation APIs to create reports from templates and various data sources.

► GroupDocs.Metadata

► GroupDocs.Search

► GroupDocs.Text

► GroupDocs.Editor



Download a Free Trial at
<https://downloads.groupdocs.com>



Americas: +1 903 306 1676

EMEA: +44 141 628 8900

Oceania: +61 2 8006 6987

sales@asposeptyltd.com

with your stakeholders, for example security, operations and release management to take your CI/CD pipeline to the next level.

Polishing the Generated Pipeline

The CI/CD pipeline is an essential element of DevOps that helps teams consistently and continuously deliver value, at a faster pace and with lower risk. You can enhance the generated pipeline to align with your process and organizational policies. You can fully automate the testing, validation, and delivery of your software in multiple environments in production, or set up a semi-automated pipeline with approvals and gates. You may even want to raise the level of quality, security and progressive exposure of new features.

Here are a few pointers to get you started:

- **Greenlighting and gates:** Refine your release pipeline with a set of gates in pre- and post-deployment options that integrate signals from monitoring systems and other external services (bit.ly/2E549uG).
- **Analyze open source projects:** Continuously analyze and measure technical quality, with SonarCloud and VSTS, from your project down to each method (bit.ly/2J4v0ea).
- **Security validation:** Continuously secure solutions within the CI/CD Pipeline. Address secure infrastructure, validate security, scan open source components for vulnerabilities, and monitor for attacks (bit.ly/2Gmczjg).
- **New release deployment:** Use deployment rings to progressively expose a new release, and fine-tune releases in production with feature flags (red.ht/2GTsQNC).

The Importance of Package Management

If you're thinking about ways to deliver more value more quickly to your end users, then you're probably using packages (such as NuGet, npm and the like). Packages and DevOps have a symbiotic relationship. Package management is focused on delivering additional value through packaged products to the CI/CD pipeline and empowering build and release automation. In VSTS, the Package Management extension is designed to be a seamless part of your DevOps pipeline, making it easier to responsibly use packages from public sources, and faster to create and share packages of your own.

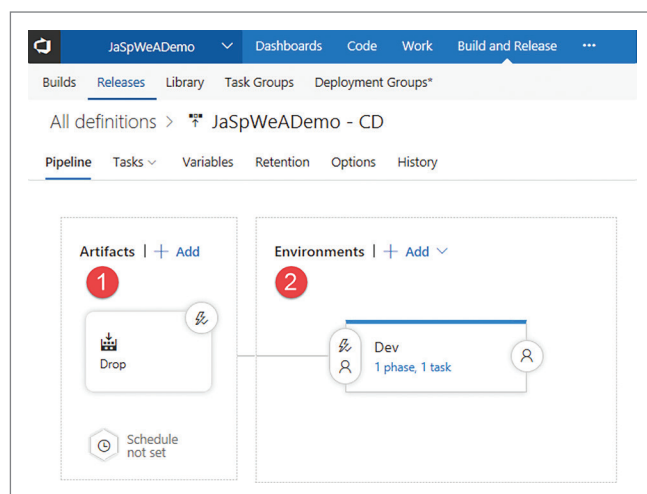


Figure 5 Pipeline View on Visual Studio Team Services

The easiest way to add some Package Management magic to your DevOps pipeline is with *upstream sources*, which connect your Package Management feed to public sources like nuget.org and npmjs.com. There are a couple big advantages to using packages from these sources through Package Management:

- Every time you use a package, a saved copy is kept in your feed, which means you're protected from incidents like the left-pad imbroglio (see bit.ly/2p01D05), public source outages and whatever else the world may throw at you. All you need to keep your pipeline flowing is your VSTS package feed.
- In your feed, you can see all the saved copies and their *provenance*, so you can filter and see exactly which packages you've used from where.

DevOps isn't a destination, it's a journey of continuous, rapid improvement.

Starting with package management takes just two steps. First, install Package Management and create a feed, as shown at bit.ly/2GDj5W9. By default, the new feed is automatically set up with upstream sources for nuget.org and npmjs.com. Then, configure Visual Studio (bit.ly/2pR3aZZ) or npm (bit.ly/2J2xVUq) to use your new feed. From there, it's business as usual. When getting started, you can run a clean build to force your packages to be saved into the feed.

Wrapping Up

Look for the Web version of this article at msdn.com/magazine/mt846654, with additional information and infographics. And expect additional coverage as we explore new and future features. We believe that transparency helps build empathy and trust, and more important, enables you to take incremental steps to adopt products and improve your process. Bookmark our timeline (bit.ly/2uv25fM), watch for future articles, and look for announcements at the upcoming Microsoft Build and Inspire events. At this point, we can mention that we're working on a few interesting enhancements for DevOps Projects, including expanding Azure VM Resources, adding support for Go and Ruby, and extending support for databases.

Now that we've introduced DevOps Projects and covered the magic behind the scenes, you should feel confident exploring ways to enhance your CI/CD pipelines. Remember, DevOps isn't a destination, it's a journey of continuous, rapid improvement. ■

WILLY-PETER SCHAUB is a program manager in VSTS, working at Microsoft Vancouver in beautiful British Columbia. Since the mid-80s, he's been striving for simplicity and maintainability in software engineering. You can follow him on LinkedIn at aka.ms/willysli or on Twitter: [@wpschaub](https://twitter.com/wpschaub).

ALEX MULLANS is a program manager in VSTS, making it easier and faster for teams to share and reuse code. You can follow him on Twitter: [@alexmullans](https://twitter.com/alexmullans).

THANKS to the following technical experts for reviewing this article:
ALM | DevOps Rangers, Gopinath Chigakkagari, Atul Malaviya



**STEAM
UNIVERSE**
THE WAY FORWARD FOR EDUCATION

IN PARTNERSHIP WITH

**THE
JOURNAL**

**CAMPUS
TECHNOLOGY**

YOUR NEW GO-TO RESOURCE

STEAM FOR EDUCATION



Artificial
Intelligence

Makerspaces

Computer
Science

Grants

Professional
Development

3D Printers

Internet of
Things

And MORE!

STEAMUniverse.com

Visual Studio® LIVE!

EXPERT SOLUTIONS FOR .NET DEVELOPERS

September 17-20, 2018
Renaissance

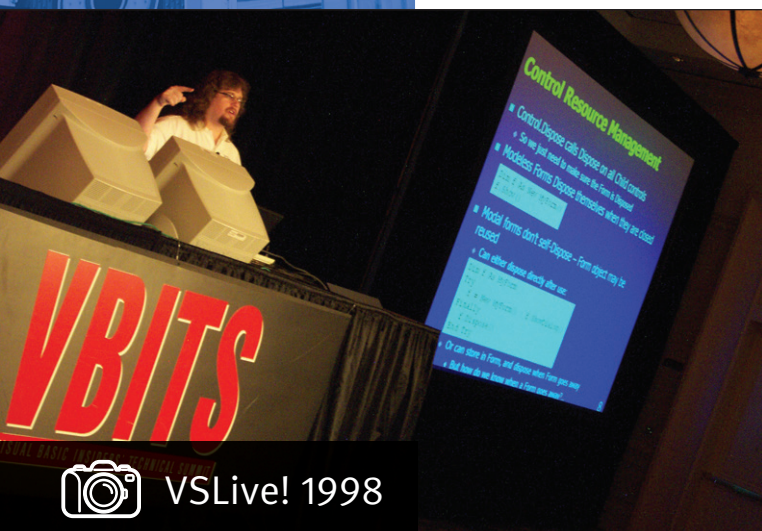
Chicago

Visual Studio LIVE! **25**
YEARS OF CODING INNOVATION
1993 - 2018

Look Back to Code Forward

Visual Studio Live! (VSLive!™) is thrilled to be returning to Chicago where developers, software architects, engineers and designers will “look back to code forward” during four days of unbiased and cutting-edge education on the Microsoft Platform.

Tackle training on the hottest topics (like .NET Core, Angular, VS2017), debate with industry and Microsoft insiders (people like Rockford Lhotka, Deborah Kurata and Brock Allen) and network with your peers—plus, help us celebrate 25 years of coding innovation as we take a fun look back at technology and training since 1993. Come experience the education, knowledge-share and networking at #VSLive25.



VSLive! 1998



VSLive! 2016

SILVER SPONSOR



SUPPORTED BY



Microsoft



Visual Studio

msdn
magazine

Visual Studio
MAGAZINE

PRODUCED BY

1105 MEDIA
YOUR GROWTH. OUR BUSINESS.

DEVELOPMENT TOPICS INCLUDE:



Visual Studio / .NET



JavaScript / Angular



Xamarin



Software Practices



Database & Analytics



ASP.NET / Web Server



ALM / DevOps



Azure / Cloud



UWP

WHAT #VSLIVE25 CAN DO FOR YOU!

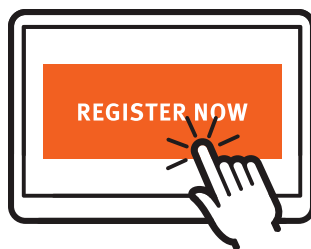


“I originally came to Visual Studio Live! to improve my technical skills, but I came back in order to network and meet other developers. The biggest change I’ve made at work, by far, is the quality of unit tests, which has resulted in a significant drop in reported bugs.”

– Justin C Kritzer, Lehigh County Government Center

“I am going to look into using ASP.NET Core as the platform for new projects as a result of attending my first VSLive! event. I also think it would be easy to move some existing projects to it. I look forward to trying the new tools and creating just what I need!”

– Abby Quick, Epiq



vslive.com/chicagomsgdn

Register to join us today.

Register Now and Save Up To \$400!

Use promo code VSLCH2

CONNECT WITH US



twitter.com/vslive –
@VSLive



facebook.com –
Search “VSLive”



linkedin.com – Join the
“Visual Studio Live” group!

vslive.com/chicagomsgdn



Under the Covers of ASP.NET Core SignalR

SignalR is the latest addition to the ASP.NET Core platform, and a long-awaited one at that. I dare say that only now with SignalR on board can we really start looking at ASP.NET Core as ready for prime time for all types of applications. No serious Web application today can do without some form of asynchronous notification and real-time capabilities.

The SignalR you know from ASP.NET has been totally rewritten and resides now in the family of ASP.NET Core libraries. In my column last month (msdn.com/magazine/mt846469), I offered a quick tour of the features and capabilities of the new ASP.NET Core SignalR. In this column, I'll dive into the internal machinery.

SignalR is an abstraction layer for bidirectional, two-way remote procedure calls (RPC) and works over a variety of transport protocols. It's host-agnostic and not limited to HTTP. In the latest version, it can transfer binary data and not just JSON-based messages. When configuring SignalR either for ASP.NET or ASP.NET Core, you can select the transport protocol and the message protocol. If you don't make an explicit choice, the transport protocol is chosen automatically and most of the time it turns out to be WebSockets. The message protocol, by contrast, is based on JSON. Let's take a look at what happens when a client—for example, a Web client—sets up a connection with a server endpoint. The code below starts a connection.

```
var progressConnection = new signalR.HubConnection("/progressDemo");
progressConnection.start();
```

If you monitor the Web traffic generated by this code with a tool like Fiddler, you'll see that two HTTP requests are sent. **Figure 1** shows the details of the first request that bootstraps the conversation.

The initial request is an HTTP POST targeted at the SignalR route specified in startup class followed by a /negotiate segment. The following code shows how to define a SignalR route:

```
app.UseSignalR(routes =>
{
    routes.MapHub<ProgressHub>("/progressDemo");
});
```

Based on the route, the initial call will target the URL: /progressdemo/negotiate.

Note that in preview versions of SignalR the OPTIONS verb was used for the same purpose. The SignalR server endpoint returns a JSON object configured as follows:

```
{
  "connectionId" : "b6668ac0-1083-499f-870a-2a5376bf5047",
  "availableTransports" : [
    "WebSockets", "ServerSentEvents", "LongPolling"
  ]
}
```

As you can see, the JSON response contains two things: the unique ID of the just-established connection, and a list of transport protocols available for use. The sample code indicates that WebSockets, ServerSentEvents and LongPolling can be used given the client and server configuration. What happens next depends on the transport actually chosen by the client.

SignalR tends to use WebSockets if possible. If not, it checks ServerSentEvents and after that falls back to LongPolling. If WebSockets can be used, then an HTTP GET request is placed to the same URL as before. This time the request also adds the connection ID as a query string parameter. The GET request is actually

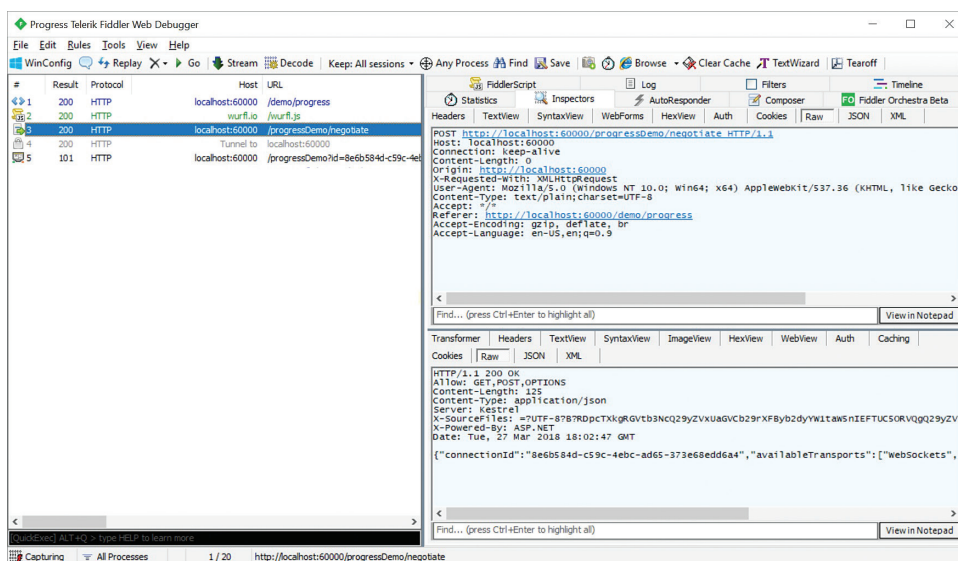


Figure 1 Details of the Initial Starter Request

Spreadsheets Everywhere.



SpreadsheetGear 2017 Released

SpreadsheetGear 2017 adds a new SpreadsheetGear for .NET Standard product, official support for Excel 2013 and Excel 2016, 51 new Excel functions for a total of 449 fully supported functions, full conditional formatting support, enhanced workbook protection and encryption, cell gradient rendering and more.



Support for iOS, Android, Linux, macOS, UWP and more

SpreadsheetGear for .NET Standard enables cross-platform developers to enjoy the same high performance Excel-compatible reporting, charting, calculations and more relied on by thousands of Windows developers for 10+ years.



Scalable Reporting

Easily create richly formatted Excel reports without Excel from any ASP.NET, Windows Forms, WPF or Silverlight application using spreadsheet technology built from the ground up for performance, scalability and reliability.



Windows
Forms



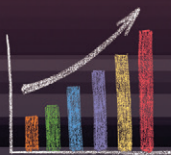
Silverlight



WPF

Powerful Controls

Add powerful Excel-compatible viewing, editing, formatting, calculating, filtering, sorting, charting, printing and more to your WinForms, WPF and Silverlight applications.



Comprehensive Charting

Enable users to visualize data with comprehensive Excel-compatible charting which makes creating, modifying, rendering and interacting with complex charts easier than ever before.



Fastest Calculations

Evaluate complex Excel-based models and business rules with the fastest and most complete Excel-compatible calculation engine available.

Download your free fully functional evaluation at SpreadsheetGear.com



a protocol upgrade request. More precisely, a protocol upgrade is a normal HTTP request (a GET but it can also be a POST) with two ad hoc headers. One is the Connection header, which must be set to Upgrade. The other is Upgrade, which must be set to the name of the required protocol—in this case, WebSockets. If the protocol upgrade is accepted, the server returns an HTTP 101 Switching Protocols response (as shown in **Figure 1**).

The client can force the connection to take place over a specific protocol.

Transport Protocols

SignalR can employ various transport protocols. The client can force the connection to take place over a specific protocol, but by default the protocol is automatically determined. To request a specific transport, simply add an extra parameter to the JavaScript code (or client code if a non-Web client is used). Here's an example:

```
var progressConnection = new signalR.HubConnection(
    "/progressDemo",
    {transport : signalR.TransportType.WebSocket});
```

Note that by passing an array instead of a direct name, you can restrict the choice to one of a few specified protocols. Of course, transport protocols differ in a few aspects. **Figure 2** lists the main characteristics of each protocol.

If the communication between the Web client and the server is cross-domain, then the server must be CORS-enabled. In this case,

Figure 2 Supported Transport Protocols

Protocol	Description
WebSockets	Based on a one-to-one connection between the client and server. Both client and server write on a shared pipe so that the flow of the data is bidirectional. The protocol cannot be used everywhere and is limited by the browser and server being used for the connection. On the client, a modern browser is required. The latest versions of all common browsers usually work, with the exception of Internet Explorer prior to version 10. On the server side, when using IIS or HttpSysServer, Windows 8 or newer is required on the client.
ServerSentEvents	Based on EventSource object active on the server. The object represents a pipe connecting the server to the client. Once the connection is established, the server can continuously send events while the client can communicate only via plain AJAX calls. The protocol dates back to the early days of Netscape and isn't supported on the Internet Explorer or Edge browsers.
LongPolling	The protocol works by opening a connection with the server to use for future responses. The connection remains pending until a response is sent or the request times out. In any case, once the connection is closed, the client immediately re-establishes it so that polling is continuous but traffic is limited to what's strictly necessary. This protocol works with all versions of all browsers and is considered a fallback solution.

you can use any available protocol (note that JSONP isn't supported in ASP.NET Core SignalR):

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
}
```

The browser adds the Origin header to the AJAX and WebSockets calls of ASP.NET Core SignalR. In addition, you need to have the CORS middleware configured in your SignalR application to ensure that the browser allows the requests.

Message Protocols

SignalR in ASP.NET always serialized exchanged messages using the text-based JSON format. The newest ASP.NET Core SignalR adds a binary message protocol based on the MessagePack format. This binary serialization format lets you exchange data in a language-agnostic way, much the way JSON does.

MessagePack is both more compact and faster to transfer than JSON, and features packet sizes that are even more compact than Binary JSON (BSON)—the MongoDB-optimized flavor of JSON. With MessagePack, both small integers and NULL values consume just one byte of data. By comparison, JSON Nulls consumes 4 bytes. For more information about the protocol, check out msgpack.org.

To use MessagePack from an ASP.NET Core SignalR Web client, you just add one more parameter to the JavaScript code that sets up the connection, as shown here:

```
var protocol = new signalR.protocols.msgpack.MessagePackHubProtocol();
var progressConnection = new signalR.HubConnection(
    "/progressDemo",
    {
        transport : signalR.TransportType.WebSocket,
        protocol : protocol
    }
);
```

Note that in the case of a Web client willing to use MessagePack, you also must include a separate JavaScript file (`signalr-msgpack-protocol.min.js`) that you find in the `@aspnet/signalr` NPM package. MessagePack also requires the ASP.NET Core server to be configured, like so:

```
public void ConfigureServices(IServiceCollection services)
{
    // SignalR already configured. Just add this:
    services.AddMessagePackProtocol();
}
```

If you use a non-Web client, all you need to do to enable MessagePack is an extra call in the client application. Specifically, you place a call to the extension method `WithMessagePackProtocol` defined on the class `HubConnectionBuilder`.

Non-Web Clients

The most interesting aspect of the .NET Standard specification is that you can consume the same library from within a variety of client applications as long as API compatibility exists. The ASP.NET Core SignalR client library, being based on the .NET Standard 2.0, can be used from within any client applications compiled for a variety of compatible platforms, including the Microsoft .NET Framework 4.6.1 and newer. This means that, say, a Windows Forms application compiled against versions of the .NET Framework newer than 4.6 can happily consume the services of an ASP.NET Core SignalR hub.

Figure 3 The Rewritten SignalR Back End

```
_connection.On("initProgressBar", () =>
{
    // Run on the UI thread
    Invoke((Action)() => label1.Text = "0%");
});
_connection.On<int>("updateProgressBar", (perc) =>
{
    // Run on the UI thread
    Invoke((Action)() => label1.Text = String.Format("{0}%", perc));
});
_connection.On("clearProgressBar", () =>
{
    // Run on the UI thread
    Invoke((Action)() =>
    {
        label1.Text = "100%";
        button1.Enabled = true;
    });
});
await _connection.StartAsync();
```

With that in mind, let's see how to launch the lengthy task discussed in last month's column and monitor it from within a new Windows Forms application. To get started, after creating the application skeleton, reference the NuGet package named `Microsoft.AspNet.Core.SignalR.Client` and all of its dependencies, like so:

```
private static HubConnection _connection;
private async void Form1_Load(object sender, EventArgs e)
{
    _connection = new HubConnectionBuilder()
        .WithUrl("http://localhost:60000/progressdemo")
        .Build();
    await _connection.StartAsync();
}
```

The code runs when the form loads up and establishes the connection with the specified SignalR endpoint. If you intend for exchanged data to be serialized using the MessagePack protocol, add one more line to the configuration code of the connection builder object, as follows:

```
private async void Form1_Load(object sender, EventArgs e)
{
    _connection = new HubConnectionBuilder()
        .WithUrl("http://localhost:60000/progressdemo")
        .WithMessagePackProtocol()
        .Build();
    await _connection.StartAsync();
}
```

The connection object you receive must be further configured with the client-side handlers responsible for refreshing the UI once notifications from the ASP.NET Core SignalR endpoint come back. Here's that code:

```
_connection.On<int>("updateProgressBar", (perc) =>
{
    this.Invoke(
        (Action)() => label1.Text = String.Format("{0}%", perc))
});
```

In the sample code, when the `updateProgressBar` notification is received, a text label is updated with the received value representing

the percentage of work currently done. You can have as many handlers as you need and the server-side SignalR counterpart exposes. **Figure 3** shows the full rewrite of the last month column's SignalR back end, as a Windows Forms client would consume it.

There are a couple of glitches you should be aware of when writing non-Web SignalR clients. First, the client handlers must be fully configured when the connection starts. Specifically, this means that the call to `StartAsync` should occur after all the due calls to the method `On<T>` have been made.

Second, keep in mind that you don't want to run the server-side—and possibly lengthy—operation that SignalR will notify on the same Windows UI thread. That would make the client application unresponsive. To work around the issue, you have to launch the server-side operation on another thread, shown here:

```
private void button1_Click(object sender, EventArgs e)
{
    Task.Run(() =>
    {
        var client = new WebClient();
        client.UploadString("http://localhost:60000/task/lengthy");
    });
}
```

Subsequently, when the server-side SignalR hub notifies back, any due changes must be conveyed to the main UI thread before they can take place. You achieve this by using the `Invoke` method in the client handler. The `Invoke` method gets a delegate and runs it on the UI thread, as shown here:

```
Invoke((Action)() =>
{
    label1.Text = String.Format("{0}%", perc);
}));
```

Figure 4 shows the sample Windows Forms application in action with a label progressively updated as the server-side operation makes any progress.

The Windows Forms application can receive notifications through any of the supported approaches: broadcast, direct connection, groups, single user and streaming. I'll have more to say about this in a future column.

Wrapping Up

ASP.NET Core SignalR supports the same transport protocols as the previous ASP.NET version, including WebSockets, ServerSentEvents and LongPolling. Furthermore, it supports a binary message protocol in addition to the canonical JSON format.

Like its predecessor, ASP.NET Core SignalR can be called from a variety of different clients—including old-fashioned Windows Forms applications. The key to achieving broad compatibility is support for the .NET Standard 2.0 specification. As we've seen in the article, if the client isn't a .NET Core application, it must be compiled to a version of the .NET Framework that's compatible with the latest standard.

The minimum version required is .NET Framework 4.6.1.

Be sure to check out the source code for this article, which can be found at bit.ly/2FxCKTs. ■

DINO ESPOSITO has authored more than 20 books and 1,000 articles in his 25-year career. Author of "The Sabbatical Break," a theatrical-style show, Esposito is busy writing software for a greener world as the digital strategist at BaxEnergy. Follow him on Twitter: @despos.

THANKS to the following Microsoft expert for reviewing this article:
Andrew Stanton-Nurse

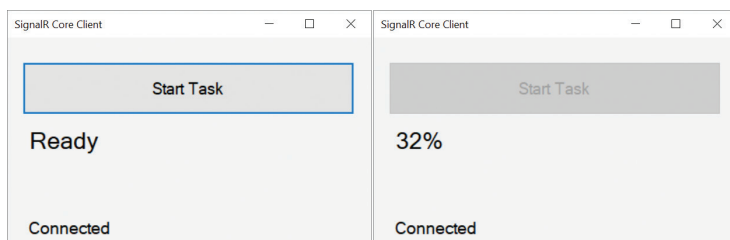


Figure 4 The Windows Forms Client Application Updated by an ASP.NET Core SignalR Hub

Visual Studio **LIVE!**

EXPERT SOLUTIONS FOR .NET DEVELOPERS

August 13 – 17, 2018
Redmond, WA

Microsoft Headquarters



Yesterday's Knowledge; Tomorrow's Code!

Visual Studio Live! (VSLive!™) is celebrating 25 years of coding innovation in 2018! From August 13 – 17, developers, software architects, engineers, designers and more will come together at Microsoft Headquarters for 5 days of unbiased education on the Microsoft Platform. Hone your skills in **Visual Studio, ASP.NET Core, AngularJS, SQL Server**, and so much more. Plus, you can eat lunch with the Blue Badges, rub elbows with Microsoft insiders, explore the campus, all while expanding your ability to create better apps!

DEVELOPMENT TOPICS INCLUDE:



Visual Studio / .NET



Xamarin



Angular / JavaScript



Azure / Cloud



Software Practices



ASP.NET / Web Server



ALM / DevOps



Database & Analytics



UWP (Windows)

EVENT SPONSOR



PLATINUM SPONSOR



SILVER SPONSOR



SUPPORTED BY



PRODUCED BY



JOIN US INSIDE THE TECH MECCA: MICROSOFT HQ!

Visual Studio Live! Redmond is back at Microsoft Headquarters this August, where you can experience the high-tech features of this state-of-the-art facility while you network with peers, code with experts and enjoy some additional unique perks with us in Redmond, including:

- **More sessions from Microsoft insiders** who are “in the trenches”, working with developers just like you.
- Visiting the **Microsoft Company Store and Visitor Center**, where you’ll have access to the employee-only discount area. You won’t want to miss out on this great opportunity!
- As part of your registration, you’ll have two chances to have **lunch in the Microsoft employee food court**, AKA The Mixer, with endless restaurant options!



“I return to Visual Studio Live! year after year (this is my third!) to see not only what bleeding edge technologies are being introduced, but to also see which technologies are fading; this helps me steer my organization in the right direction for our development needs. So many ideas gleaned from VSLive!, from Web APIs talking to SPA clients, to .NET Core, have helped me greatly!”

– Dilshan Jesook, KPMG

“I am going to look into using ASP.NET Core as the platform for new projects as a result of attending my first VSLive! event. I also think it would be easy to move some existing projects to it. I look forward to trying the new tools and creating just what I need!”

– Abby Quick, Epiq



vslive.com/redmond

Register Now and Save \$400!

Register to join us today!

Use promo code VSLRED2

CONNECT WITH US



[@vslive](https://twitter.com/vslive)



[facebook.com – Search “VSLive”](https://facebook.com/VSLive)



linkedin.com – Join the “Visual Studio Live” group!

vslive.com/redmond



Overmind

If you read the Editor's Note in last month's *MSDN Magazine*, you know that April was the 100th issue of this column. To begin my second century in this space, I've decided to tackle nothing more nor less profound than the final destination of the human species, and your and my roles in it.

I've written many times about the complete reshaping of society within the last decade due to the ubiquity of smartphones. People can't put them down (February 2012, msdn.com/magazine/hh781031). I feel mine ringing even when it isn't (November 2016, msdn.com/magazine/mt788629). They're even guiding human evolution by killing off the users too stupid to refrain from taking selfies in front of speeding trains (September 2015, msdn.com/magazine/mt422590). Kids today are practically born with them in hand, using them at the earliest of ages, while their brains are still plastic (June 2015, msdn.com/magazine/mt147245). This last generation I named digital symbionts, and I wondered where this early influence was going to take us.

Here's my first data point, from David M. Markowitz and Jeffrey T. Hancock (see slate.me/2uacSfa). They studied patients undergoing surgery with a regional anesthetic, allowing some subjects to access their phones in the recovery room, where it's usually forbidden. These patients felt better when they could use their phones. No surprise there—I like to play Pink Floyd's "Comfortably Numb" while I'm getting Novocain at the dentist.

But it's not just having the phones that made the difference, it's what the users did with them. As the authors note (my emphasis added): "The patients who could not use their phones were six times more likely to require powerful opioids to get through the procedure than those who could communicate by text message with another person. And this wasn't simply about distraction.

Patients using the phones to communicate needed fewer opioids than patients playing Angry Birds."

We humans are social animals. Being ostracized, cut off from the tribe, is the ultimate punishment. I initially regarded someone using a phone as cutting themselves off from humanity, as they ignore the people physically near them. Now I see these devices as facilitating the tighter connection of humanity—connecting our consciousnesses into a critical mass, like atoms in a sphere of imploding plutonium. Where will this end?

For a glimpse at our future, I look to Arthur C. Clarke's 1953 masterpiece, "Childhood's End"—the best science fiction novel ever written, or that ever will be. I read it to Annabelle when she was 11 years old, at the end of her childhood, the last book we read

together. (Spoiler alert: If you haven't read it yet, put this magazine down *right now*, buy it online and read it. I mean it.)

In the not-terribly-distant future (Clarke set it in the mid-1970s), Earth is taken over by the Overlords, a technologically advanced species from a distant star. They disarm humankind, preventing us from wiping ourselves out with nuclear weapons. After about 150 years of their benign dictatorship, humanity evolves, rather suddenly, into a powerful group mind. We discover that this change was what the Overlords came to Earth to facilitate. But the change only affects children younger than 10. As their minds coalesce and gain exponential power, their bodies have to be segregated even from their own parents, for the protection of both. The poignancy of the scene, as the no-longer-children enter the Overlords' ships, leaving the remainder of humanity bereft—I can't touch it. You'll have to read it yourself.

As Overlord Supervisor Karellen says, in his last speech to humanity: "... [T]hey will not possess minds as you know them. They will be a single entity, as you yourselves are the sums of your myriad cells. You will not think them human, and you will be right."

You probably aren't considering this evolution as you write your mobile apps—I didn't when I wrote my commuter rail schedule, or even Zak's mother's weight tracker. But where is this all leading? Here's Karellen's conclusion, to the unchanged, unchangeable, grieving adults:

"... [W]hat you will have brought into the world may be utterly alien, it may share none of your desires or hopes, it may look upon your greatest achievements as childish toys—yet it is something wonderful, and you will have created it."

Have we started this, my friends, you and I?

While we wait for the singularity, there are things to be done. You can find me later this month at Microsoft Build, where I'd love to hear your thoughts on life, the industry and this column. And if you happen to be in Greece on June 11, I'm keynoting the DEVit conference in Thessaloniki (devitconf.org). The topic, naturally, is the debut of my upcoming book, "Why Software Still Sucks." ■

DAVID S. PLATT teaches programming .NET at Harvard University Extension School and at companies all over the world. He's the author of 11 programming books, including "Why Software Sucks" (Addison-Wesley Professional, 2006) and "Introducing Microsoft .NET" (Microsoft Press, 2002). Microsoft named him a Software Legend in 2002. He wonders whether he should tape down two of his daughter's fingers so she learns how to count in octal. You can contact him at rollthunder.com.

Learn, Explore, Use

Your Destination for Data Cleansing & Enrichment APIs



Your centralized portal to discover our tools, code snippets and examples.

RAPID APPLICATION DEVELOPMENT

Convenient access to Melissa APIs to solve problems with ease and scalability.

REAL-TIME & BATCH PROCESSING

Ideal for web forms and call center applications, plus batch processing for database cleanup.

TRY OR BUY

Easy payment options to free funds for core business operations.

FLEXIBLE CLOUD APIS

Supports REST, JSON, XML and SOAP for easy integration into your application.

Turn Data into Success – Start Developing Today!

Melissa.com/developer

1-800-MELISSA

melissa

Introducing GrapeCity Documents, a new product line of document APIs



Documents

Complete
Document APIs
do exist!

Take total control of your digital documents with this NEW collection of ultra-fast, low-footprint document APIs for .NET Standard 2.0

These intuitive, extensible APIs allow you to create, load, modify, and save Excel spreadsheets and PDFs in any .NET Standard 2.0 application.



Expand the reach of modern apps

With full support for .NET Standard 2.0, you can target multiple platforms, devices, and cloud with one code base.



High-speed, small footprint

The API architecture is designed to generate large, optimized documents, fast—while remaining lightweight and extensible.



Comprehensive, highly programmable

Do more with your Excel spreadsheets and PDFs: these APIs support Windows, Mac, Linux, and a wide variety of features for your documents.



No dependencies

Generate and edit digital documents with no Acrobat or Excel dependencies.

GrapeCity Documents for PDF
GrapeCity Documents for Excel

Get the free trial today!
www.grapecity.com/en/documents-api

For more information
1-800-831-9006